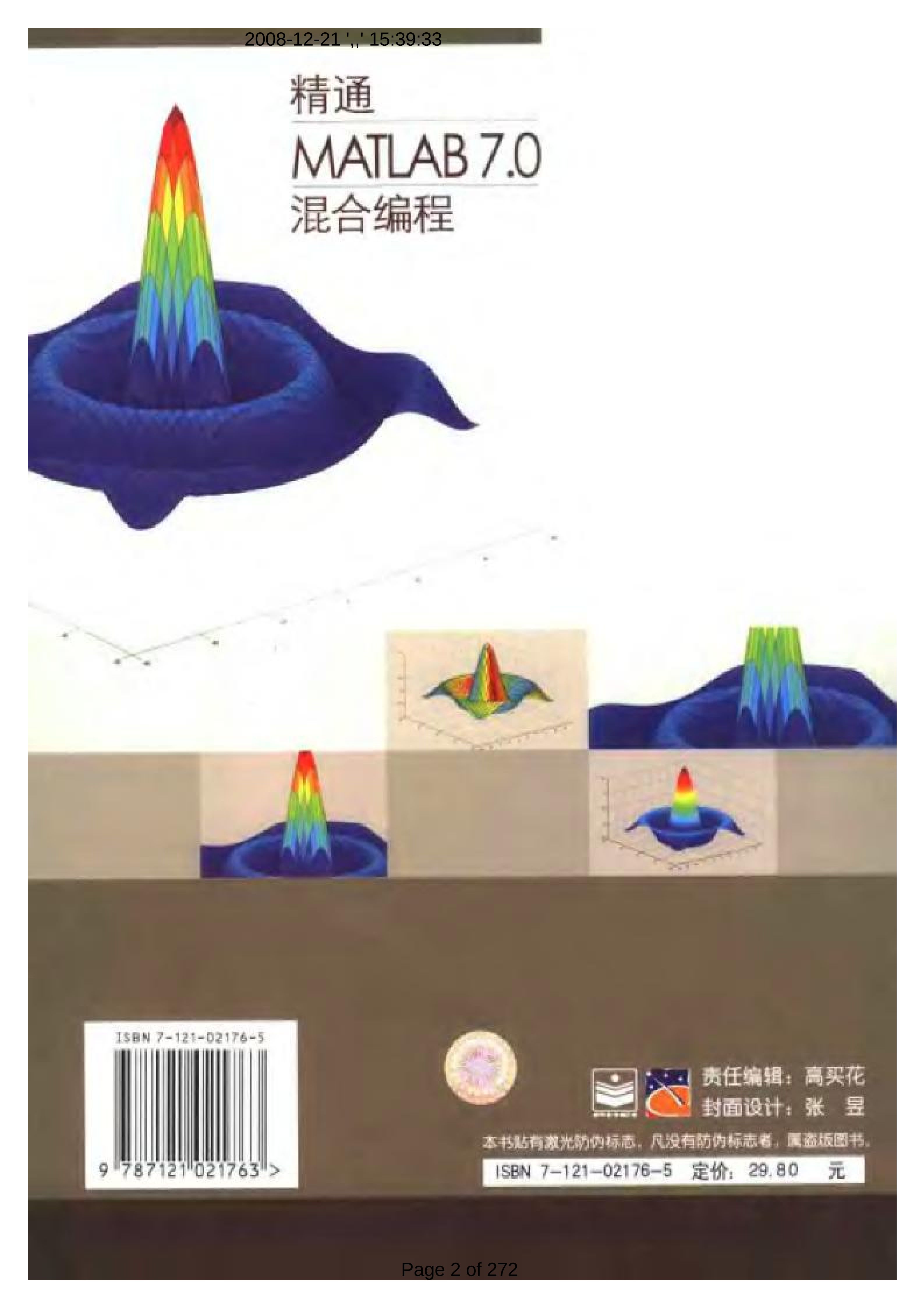
http://www.phei.com.cn

精通 MATLAB 7.0 混合编程

杨高波　亓波　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY


精通
MATLAB 7.0
混合编程
ISBN 7-121-02176-5
9 787121 021763 >
责任编辑：高买花
封面设计：张 昱
本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。
ISBN 7-121-02176-5 定价：29.80 元

精通 MATLAB 7.0 混合编程

杨高波　亓　波　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书系统地介绍 MATLAB 7.0 的混合编程方法和技巧。全书共分为 13 章。第 1 章和第 2 章介绍 MATLAB 的基础知识，第 3 章简要介绍 MATLAB 混合编程，第 4 章至第 9 章分别介绍几种典型的混合编程方法，包括 C-MEX、MATLAB 引擎、MAT 数据文件共享、Mideva、Matrix<LIB>和 Add-in。第 10 章、第 11 章介绍 MATLAB 与 Delphi 和 Excel 的混合编程。第 12 章介绍 MATLAB COM Builder，第 13 章以图像处理为例介绍了一个综合应用实例。

本书按混合编程的具体方法进行逻辑编排，自始至终用实例描述，每章着重阐述各种混合编程方法的实质和要点，同时穿插了作者多年使用 MATLAB 的经验和体会。本书既适合初学者自学，也适用于高级 MATLAB 用户，可作为高等数学、计算机、电子工程、数值分析、信息工程等课程的教学参考书，也可供上述领域的科研工作者参考。

本书所附光盘内容详尽、实例丰富，包含 MATLAB 实例的源文件、函数/命令和注解以及程序实例。

图书在版编目（CIP）数据

精通 MATLAB 7.0 混合编程 / 杨高波，亓波编著. —北京：电子工业出版社，2006.1

ISBN 7-121-02176-5

Ⅰ. 精… Ⅱ. ① 杨… ② 亓… Ⅲ. 计算机辅助计算—软件包，MATLAB 7.0 Ⅳ. TP391.75

中国版本图书馆 CIP 数据核字（2005）第 156533 号

责任编辑：高买花　　特约编辑：陈宁辉
印　　刷：北京市海淀区四季青印刷厂
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　邮编　100036
经　　销：各地新华书店
开　　本：787×1 092　1/16　印张：17　字数：435 千字
印　　次：2006 年 1 月第 1 次印刷
印　　数：5 000 册　定价：29.80 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前 言

MATLAB 是美国 MathWorks 公司开发的高性能的科学与工程计算软件。它在数值计算、自动控制、信号处理、神经网络、优化计算、小波分析、图像处理等领域有着广泛的用途。近年来，MATLAB 在国内高等院校、科研院所的应用逐渐普及，成为广大科研、工程技术人员必备的工具之一。

MATLAB 具有矩阵和数组运算方便、编程效率极高、易学易用、可扩充性强和移植性好等优点，俗称为“草稿纸式的科学计算语言”。它把工程技术人员从烦琐的程序代码中解放出来，可以快速地验证自己的模型和算法。然而，MATLAB 作为一种解释性语言，与 C 语言等其他高级语言相比较，具有不同于其他语言的特点。尽管 MATLAB 的版本不断升级，功能不断增加，但还是存在着以下一些缺点：

- 运行效率较低，执行相同功能的代码运行时间较长；
- M 文件为文本文件，文本编辑器可直接打开，不利于算法保密；
- 访问硬件能力相对较差，图形用户界面功能也不够灵活。

MATLAB 的上述缺点限制了它不能作为通用的软件开发平台。常见的通用编程平台（如 Visual C++、Visual Basic、Delphi 和 Fortran 等）功能强大且灵活，但编程效率较低，尤其是当需要快速验证算法时。因此，在实际的工程应用中，需要发挥 MATLAB 和其他高级语言各自的优势，以降低开发难度，缩短编程时间。幸运的是，MATLAB 的应用程序接口（API）为实现 MATLAB 与 Visual C++等通用编程平台的混合编程提供了可能。MATLAB Compiler（编译器）提供了将 MATLAB 语言编写的 M 文件自动转换为 C 或 C++文件的能力，支持用户进行独立的应用开发。结合 MathWorks 提供的 C/C++数学库和图形库，用户可以利用 MATLAB 快速地开发出功能强大的独立应用。

本书针对最新的版本 MATLAB 7.0，介绍它与其他高级语言的混合编程技术。全书共分为 13 章和 2 个附录：

第 1 章 MATLAB 概述，主要介绍 MATLAB 的发展历程、语言特点及 MATLAB 7.0 版本相对于以前版本所具有的一些新功能。

第 2 章 MATLAB 程序设计及代码优化，简要介绍 MATLAB 的变量和表达式、流程控制、编程规范等程序设计基础知识，还对编程过程中提高 M 文件执行效率的一些技术进行了详细的介绍。

第 1 章、第 2 章的主要目的是帮助 MATLAB 初学者对 MATLAB 有一个整体的了解，用过 MATLAB 的读者可以跳过这两章。但是，第 2.8 节“提高 M 文件执行效率的技巧”具有较强的工程实际意义，建议读者阅读。

第 3 章对 MATLAB 的混合编码进行了简要介绍，包括混合编码的出发点、分类，以及几种常见的混合编程方法。

第 4 章对 C-MEX 进行了介绍，包括 MEX 文件系统的配置、结构、运行、Visual C++ 环境下 MEX 文件的调试和实例。

第 5 章介绍通过 MATLAB 引擎（Engine）实现混合编程，包括引擎库函数、环境配置、引擎类的封装及实例。

第 6 章对通过 MAT 文件实现 MATLAB 和其他应用程序的数据共享进行了介绍，包括 MAT 文件简介、操作 MAT 文件的 MATLAB API 函数、Visual C++调用 MAT 时的环境设置以及实例。

第 7 章介绍通过 Mideva 实现混合编程的方法，包括 Mideva 安装、Mideva 环境下 M 文件向 EXE 或 DLL 文件的转化、Visual C++环境下通过 Mideva 实现混合编程的环境配置方法及实例。

第 8 章对利用 Matrix<LIB>数学函数库实现混合编程进行了介绍。

第 9 章介绍利用 MATLAB 6.x 版本特有的 Add-in 插件实现混合编程，包括安装、环境配置以及应用实例。

第 10 章对 MATLAB 和 Delphi 的混合编程进行了介绍，包括自动化（ActiveX）、引擎、动态链接库以及外部调用等方式。

第 11 章简要介绍 MATLAB 和 Excel 的混合编程，包括 Excel Link 和 MATLAB Builder for Excel 两种方式。

第 12 章介绍 MATLAB COM Builder，包括它的安装、C/C++编译器的配置以及在 Visual C++中使用 MATLAB COM Builder 生成的组件实例。

第 13 章介绍综合应用实例，以一个简易的图像处理工具软件为例综合本书的知识。

附录 A 和附录 B 中分别列举了常见的免费 MATLAB 工具箱以及 MATLAB 站点。

本书第 3～9 章以及第 11 章和第 12 章由杨高波编写，第 1 章、第 2 章、第 10 章和第 13 章由亓波编写，由杨高波统稿。全书由湖南大学计算机与通信学院李仁发教授主审。在本书的编写过程中，陈薇薇、雷靖和李清畅等硕士生协助完成了文字核对工作，在此表示衷心的感谢。

本书编著者是国内 MATLAB 最早的用户之一，1998 年就开始在科研工作中使用，特别是在《计算机世界》上发表了几篇关于 MATLAB 混合编程的论文后，经常有网友通过邮件咨询此方面的知识，本书也正是在他们的要求和鼓励下写成的。只要认真阅读并在计算机上逐个调试书中的例程(本书所有例程的源代码都可以在附带的光盘中找到，按章节划分目录存储)，读者终会有所收获。由于编著者水平有限，错误和不足之处在所难免，敬请读者多提宝贵意见，以便继续完善，共同进步。如果您对书的内容有任何疑问，请与编著者讨论或批评指正。联系方式如下：

杨高波　gbyang_hunu@hotmail.com

亓　波　dvdsoft78@msn.com

编著者

2005 年 10 月

目　录

第 1 章　MATLAB 概述 ………………………………………………………… (1)

1.1　MATLAB 的发展历程 ………………………………………………… (1)

1.2　MATLAB 产品组成及语言特点 ……………………………………… (2)

1.2.1　MATLAB 的主要产品构成 ……………………………………… (2)

1.2.2　MATLAB 语言的特点 …………………………………………… (4)

1.3　MATLAB 7.0 的新功能和新产品 ……………………………………… (5)

1.3.1　MATLAB 7.0 的新功能 …………………………………………… (5)

1.3.2　MATLAB 升级及新增的模块 …………………………………… (7)

1.4　小结 …………………………………………………………………… (11)

第 2 章　MATLAB 程序设计及代码优化 ………………………………… (12)

2.1　MATLAB 的表达式和变量 …………………………………………… (12)

2.1.1　表达式 ……………………………………………………………… (12)

2.1.2　变量 ………………………………………………………………… (12)

2.2　细胞数组与结构数组 ………………………………………………… (12)

2.2.1　细胞数组 …………………………………………………………… (12)

2.2.2　结构数组 …………………………………………………………… (13)

2.3　类与对象 ……………………………………………………………… (13)

2.4　流程控制 ……………………………………………………………… (17)

2.4.1　for 循环结构 ……………………………………………………… (17)

2.4.2　while 循环结构 …………………………………………………… (18)

2.4.3　if-else-end 分支结构 ……………………………………………… (18)

2.4.4　switch-case 结构 ………………………………………………… (18)

2.4.5　try-catch 结构 ……………………………………………………… (19)

2.5　M 文件编程 …………………………………………………………… (19)

2.6　M 文件编程规范 ……………………………………………………… (22)

2.7　M 文件评述器 ………………………………………………………… (30)

2.8　提高 M 文件执行效率的技巧 ………………………………………… (32)

2.8.1　矢量化操作 ………………………………………………………… (33)

2.8.2　给数组预定义维 …………………………………………………… (34)

2.8.3　下标或者索引操作 ………………………………………………… (35)

2.8.4　尽量多使用函数文件而少使用非脚本文件 ……………………… (35)

2.8.5　将循环体中的内容转换为 C-MEX ……………………………… (35)

2.8.6　内存优化 …………………………………………………………… (35)

2.9　小结 …………………………………………………………………… (36)

第 3 章　MATLAB 混合编程简介 …… (37)

3.1　进行混合编程的出发点 …… (37)
3.2　MATLAB 应用程序接口简介 …… (37)
3.3　几种常见的混合编程方法简介 …… (39)
3.3.1　使用 MATLAB 自带的 MATLAB Compiler …… (39)
3.3.2　利用 MATLAB 引擎 …… (40)
3.3.3　利用 ActiveX 控件 …… (40)
3.3.4　利用 MAT 文件 …… (41)
3.3.5　C-MEX …… (41)
3.3.6　利用 Mideva/Matcom …… (41)
3.3.7　利用 Matrix<LIB>实现混合编程 …… (42)
3.3.8　利用 MATLAB Add-in …… (42)
3.3.9　MATLAB COM Builder …… (42)
3.3.10　MATLAB 和 Excel 混合编程 …… (43)
3.4　小结 …… (44)

第 4 章　C-MEX 编程 …… (45)

4.1　C-MEX 简介 …… (45)
4.2　MEX 文件系统的配置 …… (45)
4.3　MEX 文件的结构和运行 …… (46)
4.3.1　MEX 文件结构 …… (46)
4.3.2　MEX 函数的执行流程 …… (49)
4.3.3　MEX 文件的结构和使用 …… (50)
4.3.4　MEX 文件与独立应用程序的区别 …… (50)
4.4　C 语言 MEX 函数 …… (51)
4.5　C-MEX 混合编程 …… (54)
4.6　Visual C++中 MEX 文件的建立和调试 …… (55)
4.6.1　Visual C++中 MEX 程序的建立和环境设置 …… (57)
4.6.2　MEX 程序的调试 …… (59)
4.6.3　MEX 独立应用程序的发布 …… (61)
4.7　MEX 编程实例 …… (61)
4.8　小结 …… (67)

第 5 章　通过 MATLAB 引擎实现混合编程 …… (68)

5.1　MATLAB 引擎简介 …… (68)
5.2　MATLAB 引擎库函数 …… (68)
5.3　Visual C++调用 MATLAB 引擎时的环境设置 …… (74)
5.4　MATLAB 引擎类的封装 …… (79)
5.4.1　CMATLABEng 类的定义和实现代码 …… (79)

5.4.2 CMATLABEng 说明 …… (82)
5.4.3 CMATLABEng 说明和使用方法 …… (84)
5.5 应用实例 …… (84)
5.6 小结 …… (87)
第 6 章 MAT 文件实现数据共享 …… (88)
6.1 MAT 文件简介 …… (88)
6.2 操作 MAT 文件 …… (88)
6.2.1 MAT 文件格式 …… (88)
6.2.2 操作 MAT 文件的 MATLAB API …… (90)
6.3 Visual C++调用 MAT 时的环境设置 …… (96)
6.4 实例 …… (97)
6.5 小结 …… (107)
第 7 章 利用 Mideva 实现混合编程 …… (108)
7.1 Mideva 简介 …… (108)
7.2 Mideva 的安装 …… (108)
7.3 Mideva 环境下 M 文件到 dll/exe 文件的转换 …… (110)
7.4 Visual C++环境下使用 Mideva 混合编程 …… (111)
7.4.1 混合编程环境的设置 …… (112)
7.4.2 通过外壳函数调用 …… (112)
7.5 Matrix<LIB> …… (113)
7.6 混合编程实例 …… (113)
7.7 小结 …… (122)
第 8 章 利用 Matrix<LIB>实现混合编程 …… (123)
8.1 Matrix<LIB>简介 …… (123)
8.2 Matrix<LIB>与 Visual C++混合编程 …… (123)
8.2.1 Matrix<LIB>的安装 …… (123)
8.2.2 Visual C++环境配置 …… (123)
8.2.3 初始化库 …… (124)
8.3 Matrix<LIB>函数使用参考 …… (124)
8.3.1 矩阵操作 …… (125)
8.3.2 库常量 …… (128)
8.3.3 访问库函数 …… (129)
8.3.4 矩阵 I/O …… (130)
8.3.5 图形函数 …… (131)
8.4 混合编程实例 …… (133)
8.5 MATLAB 数学库 …… (139)
8.5.1 简介 …… (139)
8.5.2 Visual C++ 工程中调用 MATLAB 数学函数库的环境设置 …… (140)

8.6 小结 …… (141)

第9章 通过 MATLAB Add-in 实现混合编程 …… (142)

9.1 MATLAB Add-in 简介 …… (142)
9.2 MATLAB Add-in 安装和在 Visual C++中的环境设置 …… (142)
9.3 通过 MATLAB Add-in 生成独立应用程序 …… (145)
9.4 MATLAB Add-in 实例 …… (146)
9.5 小结 …… (149)

第10章 MATLAB 和 Delphi 混合编程 …… (150)

10.1 Delphi 开发环境介绍 …… (150)
10.2 通过 MATLAB 自动化服务实现混合编程 …… (150)
10.2.1 自动化服务的实现方法 …… (150)
10.2.2 自动化服务应用举例一 …… (152)
10.2.3 自动化服务应用举例二 …… (154)
10.3 利用 MATLAB 引擎实现混合编程 …… (158)
10.3.1 动态链接库介绍 …… (158)
10.3.2 在 Delphi 中调用 Visual C++创建的动态链接库的实例 …… (158)
10.3.3 MATLAB 引擎动态链接库的设计 …… (163)
10.4 Delphi 调用 Mideva 生成的动态链接库 …… (168)
10.4.1 Mideva 介绍 …… (168)
10.4.2 应用实例 …… (168)
10.5 通过外部调用实现混合编程 …… (174)
10.5.1 外部调用方法介绍 …… (174)
10.5.2 应用实例 …… (175)
10.6 小结 …… (176)

第11章 MATLAB 和 Excel 的混合编程 …… (177)

11.1 引言 …… (177)
11.2 通过 Excel Link 实现 Excel 和 MATLAB 的数据共享 …… (177)
11.2.1 概述 …… (177)
11.2.2 Excel Link 的安装 …… (177)
11.2.3 Excel Link 的函数 …… (180)
11.2.4 Excel Link 应用实例 …… (181)
11.2.5 Excel Link 的注意事项 …… (183)
11.3 通过 Excel 生成器 …… (184)
11.3.1 概述 …… (184)
11.3.2 创建 Excel 生成器插件 …… (184)
11.4 直接将 MATLAB 工作区间的数据拷贝到 Excel …… (186)
11.5 小结 …… (188)

第 12 章　通过 MATLAB COM Builder 实现混合编程 …… (189)
12.1　COM 基础知识 …… (189)
12.2　MATLAB 支持的组件自动化 …… (190)
12.2.1　在 MATLAB 下运行其他软件的组件 …… (190)
12.2.2　在其他程序下运行 MATLAB 的组件 …… (194)
12.2.3　MATLAB COM Builder 简介 …… (195)
12.3　MATLAB COM Builder 使用 …… (195)
12.3.1　配置 MATLAB C/C++编译器 …… (195)
12.3.2　使用 MATLAB COM Builder …… (195)
12.3.3　MATLAB COM Builder 工具库 …… (198)
12.3.4　在 Visual C++中调用 COM 组件的步骤 …… (199)
12.4　在 Visual C++中使用 MATLAB COM Builder 生成的组件实例 …… (200)
12.5　小结 …… (206)
第 13 章　混合编程综合应用实例 …… (207)
13.1　引言 …… (207)
13.2　预备知识 …… (207)
13.2.1　数字图像处理简介 …… (207)
13.2.2　MATLAB 图像处理工具箱简介 …… (208)
13.2.3　Visual C++的图像处理位图文件读/写操作 …… (209)
13.3　综合实例框架 …… (210)
13.3.1　框架搭建 …… (210)
13.3.2　模块划分 …… (213)
13.3.3　应用程序功能添加 …… (227)
13.4　实现方法 …… (232)
13.4.1　图像直方图统计——MATLAB 引擎命令实现 …… (233)
13.4.2　图像形态学——MATLAB 引擎数据交互实现 …… (237)
13.4.3　图像的 FFT 变换——通过 Mideva 实现 …… (250)
13.5　小结 …… (257)
附录 A　常见的免费 MATLAB 工具箱 …… (258)
附录 B　常用的 MATLAB 免费站点 …… (259)
参考文献 …… (260)

第 1 章　MATLAB 概述

MATLAB 的名称源自 Matrix Laboratory，它是 MathWorks 公司开发的一种科学计算软件。MATLAB 将高性能的数值计算、符号计算和可视化集成在一起，并提供了大量的内置函数，从而被广泛地应用到科学计算、控制系统、信息处理等领域的分析、仿真和设计工作中。MATLAB 是国际公认的优秀数学应用软件之一。

本章介绍 MATLAB 的基础知识，包括 MATLAB 的发展历程、语言特点及 MATLAB 7.0 的新产品和新功能，帮助读者对 MATLAB 有一个整体的了解。

1.1　MATLAB 的发展历程

20 世纪 70 年代中期，数值计算成为工程技术和科研的有效手段之一。Cleve Moler 博士及其同事共同开发了基于 Fortran 语言的 LINPACK 和 EISPACK 函数库，以支持数值计算。这两个函数库代表了当时矩阵计算的最高水平，并为广大科技及工程人员所广泛使用。20 世纪 70 年代末期，Cleve Moler 博士发现在使用这两个函数库的过程中大量时间都被放在了接口程序设计上，为了减少工作量，Cleve Moler 亲自编写了 LINPACK 和 EISPACK 函数库的接口程序，并以 MATLAB 作为该接口程序的名字。

20 世纪 80 年代初期，Cleve Moler 与 John Little 等利用 C 语言开发了新一代的 MATLAB 语言，包括编译解释程序、图形功能的设计以及各类数学分析的子模块，并撰写了用户指南。1984 年，为了推广 MATLAB 在数值计算中的应用，Cleve Moler 与 John Little 等正式成立了 MathWorks 公司，从而把 MATLAB 推向市场，并开始了对 MATLAB 工具箱等的开发设计。

1993 年，MathWorks 公司推出了 MATLAB 4.0 版本，1995 年推出 MATLAB 4.2C 版本，1997 年推出了 MATLAB 5.x 版本（Release 11），2000 年推出 MATLAB 6 版本（Release 12），2003 年推出 MATLAB 6.5 版本（Release 13），最新版本是 2004 年 7 月推出的 7.0 版本（Release 14）。

新的版本集中了日常数学处理中的各种功能，包括高效的数值计算、矩阵运算、信号处理和图形生成等功能。在 MATLAB 环境下，用户可以集成地进行程序设计、数值计算、图形绘制、输入/输出、文件管理等各项操作。MATLAB 提供了一个人—机交互的数学系统环境。该系统的基本数据结构是矩阵，在生成矩阵对象时，不要求明确定义矩阵的维数。与利用 C 语言或 Fortran 语言进行数值计算的程序设计相比，利用 MATLAB 可以节省大量的编程时间。在美国的一些大学里，MATLAB 正在成为对数值线性代数，以及其他一些高等应用数学课程进行辅助教学的有益工具。在工程技术界，MATLAB 也被用来解决一些实际课题和数学模型问题。由于 MATLAB 的开放性和可移植性，自第 1 版发行以来，已有众多的科技工作者加入到 MATLAB 的开发队伍中，并为形成最新版本的 MATLAB 系统做出了巨大的贡献。

MATLAB 以商品形式出现后，仅短短几年，就以其良好的开放性和运行的可靠性，使

原先控制领域里的封闭式软件包（如英国的 UMIST，瑞典的 LUND 和 SIMNON，德国的 KEDDC）纷纷淘汰，而改以 MATLAB 为平台加以重建。在 20 世纪 90 年代，MATLAB 已经成为国际控制界公认的标准计算软件。到 90 年代初期，在国际上 30 多个数学类科技应用软件中，MATLAB 在数值计算方面独占鳌头，而 MatheMatica 和 Maple 则分居符号计算软件的第二和第三名。MathCAD 因其提供计算、图形、文字处理的统一环境而深受中学生欢迎。在欧美大学里，诸如应用代数、数理统计、自动控制、数字信号处理、模拟与数字通信、时间序列分析、动态系统仿真等课程的教科书都把 MATLAB 作为内容，MATLAB 是攻读学位的大学生、硕士生、博士生必须掌握的基本工具。因此，MATLAB 已经被确认为准确、可靠的科学计算标准软件。在许多国际一流学术刊物上（尤其是信息科学刊物），都可以看到 MATLAB 的应用。在设计研究单位和工业部门，MATLAB 被认定是进行高效算法研究和开发的首选软件工具。如美国 National Instruments 公司信号测量、分析软件 LabVIEW，Cadence 公司信号和通信分析设计软件 SPW 等，或者直接建筑在 MATLAB 之上，或者以 MATLAB 为主要支撑。又如 HP 公司的 VXI 硬件，TM 公司的 DSP，Gage 公司的各种硬卡、仪器等都接受 MATLAB 的支持。

1.2 MATLAB 产品组成及语言特点

MATLAB 支持从概念设计、算法开发、模型仿真和实时实现的理想集成环境。无论进行科学研究还是工程应用，MATLAB 都是必不可少的模型和算法仿真工具。一般而言，MATLAB 的典型应用包括：

- 数据分析和可视化；
- 数值和符号计算；
- 建模、仿真和原型开发；
- 算法预设计与验证；
- 图形用户界面设计；
- MATLAB 应用与 C、C++、Java 以及 Web 集成；
- 图像和视频信号处理；
- 一些特殊的矩阵计算应用，例如自动控制理论、统计、数字信号处理（时间序列分析）等。

1.2.1 MATLAB 的主要产品构成

MATLAB 由一组面向具体应用的工具箱组成，包含了完整的函数集用来对数字图像、控制系统、小波分析和神经网络等特殊应用进行分析和设计。MATLAB 的工具箱是开放的，因此 MATLAB 的工具箱越来越多，功能也越来越强大。用户可以编写自己的工具箱，使用时与使用 MATLAB 提供的工具箱一样。因此，出现了越来越多的免费或商业 MATLAB 工具箱（参见附录 A）。

MATLAB 的主要产品构成如下。

1. MATLAB 集成开发环境

MATLAB 提供了一个集成的开发环境，方便用户开发自己的应用程序。它有一系列的工

具和功能体，其中大部分具有图形用户界面，包括桌面（Desktop）、命令窗口、历史窗口（History）、工作空间（Workspace）、文件和搜索路径等。

2. MATLAB 数学函数库

MATLAB 提供了强大的数学函数库，既包括最基本的矩阵运算函数，如矩阵求逆等，又包括一些特殊的数学函数，如贝塞尔函数等。数学函数库是 MATLAB 进行数据分析的基础。

3. MATLAB 图形用户接口

MATLAB 提供了图形用户接口函数，包括二维和三维图形显示、图像处理、动画和图形显示的高级命令。设计图形用户界面（GUI）的工具包括布局编辑器、排列工具、属性观察器和菜单编辑器等。

4. MATLAB 的专用领域工具箱

MATLAB 提供了一系列专用领域的工具箱，如模型仿真、神经网络、小波分析、信号处理、图像处理等，用于解决特定领域的工程问题。工具箱是开放和可扩展的，用户可以根据需要选择购买和选择安装需要的工具箱。

5. MATLAB Compiler

MATLAB Compiler（编译器）提供了将 MATLAB 语言编写的 M 文件自动转换为 C 或 C++格式文件的能力，支持用户进行独立应用开发。利用 MATLAB Compiler，用户可以快速地开发出功能强大的独立应用。

6. MATLAB Simulink

Simulink 是一个对动态系统进行建模、仿真和分析的软件包。它既可以仿真线性系统，又可以仿真非线性系统。它使得 MATLAB 的功能得到了进一步的扩展：

- 实现了可视化建模，在 Windows 环境下，用户可以通过简单的鼠标操作建立直观的系统模型，进行分析仿真。
- 实现了与 MATLAB 中 M 文件的数据共享，甚至可以与硬件实现实时信息交换。
- 将理论研究与工程实际有机地结合在一起。

7. Stateflow

与 Simulink 的模型框结合，描述复杂事件驱动系统的逻辑行为，驱动系统在不同的模块之间进行切换。

8. Real-Time Workshop

Real-Time Workshop 与 Stateflow 直接从 Simulink 模型与 Stateflow 框图中生成高效的可移植 C 代码或 Ada 代码。只需要简单的操作，用户无须烦琐的手工编程与调试就可以生成应用代码。

MATLAB 安装完后，在 MATLAB 命令符下输入 ver 命令，即可了解安装了哪些工具箱以及它们的版本。本书的 MATLAB 7.0 工具箱版本如下（仅列举了部分与混合编程相关的及

个别应用工具箱）：

```
>> ver
-------------------------------------------------------------------------------------
MATLAB Version 7.0.0.19920（R14）
MATLAB License Number: 0
Operating System: Microsoft Windows 2000 Version 5.0（Build 2195: Service Pack 4）
Java VM Version: Java 1.4.2 with Sun Microsystems Inc. Java HotSpot（TM）Client VM
-------------------------------------------------------------------------------------
MATLAB                              Version 7.0        （R14）
Simulink                            Version 6.0        （R14）
Excel Link                          Version 2.2        （R14）
Image Processing Toolbox            Version 4.2        （R14）
MATLAB Builder for COM              Version 1.1        （R14）
MATLAB Builder for Excel            Version 1.2        （R14）
MATLAB Compiler                     Version 4.0        （R14）
```

1.2.2 MATLAB 语言的特点

MATLAB 具有不同于其他语言如 Fortran、C 语言等的特点，被称为第四代计算机语言，又称为“草稿纸式”的语言。MATLAB 把工程技术人员从烦琐的程序代码中解放出来，可以快速地验证自己的模型和算法。概括起来，MATLAB 语言具有如下主要特点。

1. 方便的矩阵和数组运算

MATLAB 是以矩阵为基础的，可以方便地进行矩阵的算术运算、关系运算和逻辑运算等。MATLAB 有特殊矩阵专门的库函数，可以高效地求解诸如信号处理、控制、优化等问题。变量不需要预先定义，也不需要预先定义矩阵（包括数组）的维数。

2. 编程效率极高

MATLAB 是一种面向科学和工程计算的高级语言。它以矩阵运算为基础，极少的代码即可实现复杂的功能。例如求矩阵的秩，MATLAB 只需要一条语句 det( )，而 C 语言等则需要几十甚至上百条代码。

3. 易学易用，使用方便

MATLAB 易学易用，其函数名和表达更接近我们书写计算公式的思维表达方式，MATLAB 编写程序犹如在演草纸上排列公式与求解问题。MATLAB 是一种解释性语言，不需要专门的编译器。具体地说，MATLAB 运行时，可直接在命令行输入 MATLAB 语句，系统立即进行处理，完成编译、链接和运行的全过程。因此，MATLAB 语言不仅是一门语言，广义上是一种语言调试系统。

4. 可扩充性强

MATLAB 有着丰富的库函数，在进行复杂的数学运算时可以直接调用。用户可以根据需

要方便地编写和扩充新的函数库。为了充分利用 Fortran 和 C 语言资源，用户可以通过混合编程在 MATLAB 中调用 Fortran 和 C 语言的源程序，也可以在 C 语言和 Fortran 中使用 MATLAB 的数值计算功能。

5. 可移植性好

MATLAB 本身是用 C 语言编写的，而 C 语言的可移植性好。MATLAB 函数可以很方便地移植到 C 语言平台。除了内部函数以外，MATLAB 的绝大部分函数和工具箱函数都是公开的，可以用文本编辑器打开。

但是，MATLAB 作为一种解释性语言，与 C 语言等其他高级语言相比较，也存在着以下一些缺点：

（1）运行效率较低，执行相同功能的代码运行时间较长；

（2）M 文件为文本文件，文本编辑器可直接打开，不利于算法保密；

（3）访问硬件能力相对较差，图形用户界面功能也不够灵活。

1.3 MATLAB 7.0 的新功能和新产品

1.3.1 MATLAB 7.0 的新功能

MATLAB 的版本发展较快，自 20 世纪 80 年代诞生以来已发展到 7.0 版本。2004 年 MathWorks 推出了最新的 7.0 版本，它较以前版本有了较大的改进。MATLAB 7.0 的特点在于全新的桌面及各种不同领域的集成工具，以方便用户使用。MATLAB 7.0 的桌面如图 1-1 所示。

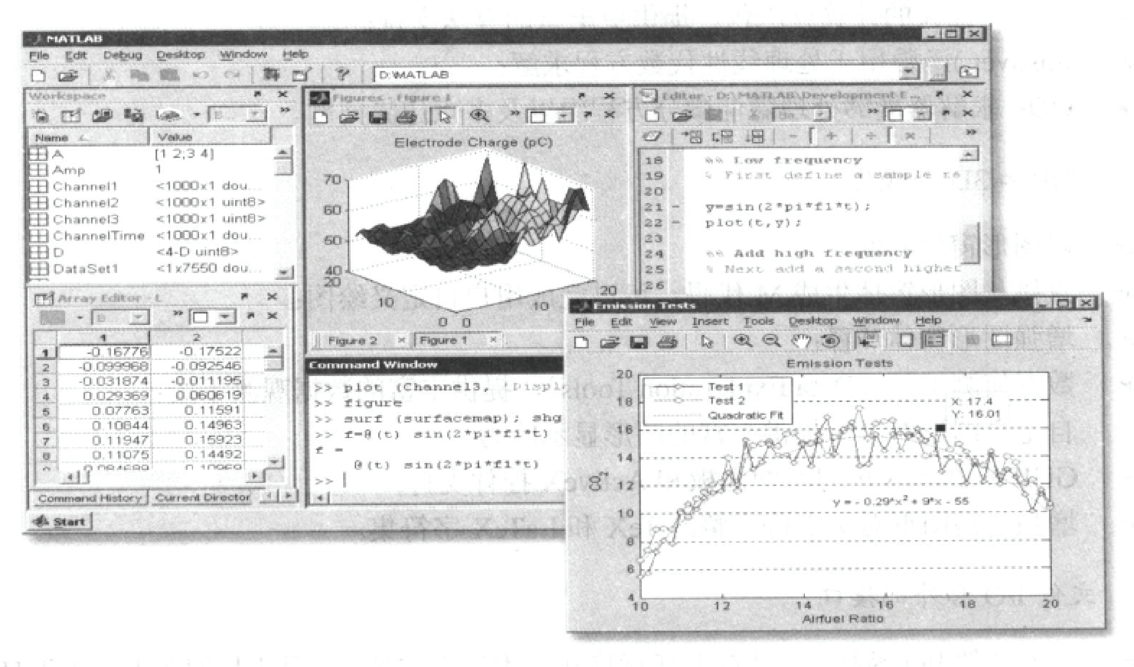


图 1-1 MATLAB 7.0 桌面

MATLAB 7.0 针对编程环境、代码效率、数据可视化、数学计算、文件 I/O 等方面进行升级，具体内容包括以下方面。

1. 开发环境

- 重新设计的桌面环境，针对多文档界面应用提供了简便的管理和访问方法，允许用户自定义桌面外观、创建常用命令的快捷方式；
- 增强数组编辑器（Array Editor）和工作空间浏览器（Workspace Brower）功能，用于数据的显示、编辑和处理；
- 在当前目录浏览器（Current Directory Browser）工具中，增加代码效率分析、覆盖度分析等功能；
- M-Lint 编码分析，辅助用户完成程序性能分析，提高程序执行效率；
- 增强 M 文件编辑器(M-Editor)，支持多种格式源代码文件可视化编辑，例如 C/C++、HTML、Java 等。

2. 编程

- 支持创建嵌套函数（Nested Function），提供更灵活的代码模块化方式；
- 匿名函数（Anonymous Function）功能，支持在命令行或者脚本文件中创建单行函数（Single Line Function）；
- 支持条件分支断点，可以在条件分支语句中进行程序中断调试；
- 模块化注释，支持为代码段注释。

3. 数学

- 支持整数算术运算；
- 支持单精度数据类型运算，包括基本算术运算、线性代数、 FFT 等；
- 使用更强大的计算算法包，提供更丰富的算法支持；
- linsove( )函数用于处理线性代数方程求解；
- ODE 求解器能够处理隐性微分方程组以及多点边界问题。

4. 图形和 3D 可视化

- 新图形窗体界面；
- 直接从图形窗体生成 M 代码，可以完成用户自定义绘图；
- 增强图形窗体注释；
- 数据侦测工具（Data Exploration Tools），提供丰富的数据观测手段；
- 自定义图形对象，提供丰富的图形显示能力；
- GUIDE 新增对用户界面面板和 ActiveX 控件支持；
- 增强句柄图形对象支持完整的 TeX 和 LaTeX 字符集。

5. 文件 I/O 和外部接口

- 新增文件 I/O 函数，支持读取任意格式文本数据文件，并且支持写入 Excel 和 HDF5 格式数据文件；
- 具有压缩功能的 MAT 文件格式，支持快速数据文件 I/O 能力；
- javaaddpath( )函数，无须重新启动 MATLAB 即完成 Java 类的加载、删除等功能；

- 支持 COM、服务器事件以及 VBS；
- 支持 SOAP，使用网络服务；
- FTP 对象，直接访问 FTP 服务器；
- 支持 Unicode 编码格式，增强 MAT 文件字符集。

6. 性能与系统平台支持

- JIT 加速器支持所有数值数据类型；
- Windows XP 系统下支持 3GB 内存访问。

1.3.2 MATLAB 升级及新增的模块

MATLAB 7.0 升级了 29 个产品模块，新增了 12 个产品模块。

1.3.2.1 模块主要特性总结

1. Control System Toolbox 6

Control System Toolbox 6 工具箱为动态系统闭环控制器设计与分析提供了强大的工具，主要新特性包括：

- 使用基于LAPACK和SLICOT算法引擎的强大数值计算引擎，提高计算速度与精度；
- 针对非稳态系统提供了更好的模型缩减算法；
- 新开发的模型分解命令。

2. Database Toolbox 3

Database Toolbox 3 支持在 MATLAB 环境中访问支持 ODBC/JDBC 标准的数据库，使用 Visual Query Builder 工具可以完成数据库的访问、修改等工作，这一过程不需工程师了解任何 SQL 语言，主要新特性包括：

- 支持 Java 语言的 SQL 对象 BINARY 和 OTHER；
- 支持脱离 Visual Query Builder 直接向 ODBC/JDBC 数据库写入数据；
- Visual Query Builder 支持结构和数值数组。

3. Filter Design Toolbox 3

Filter Design Toolbox 3 支持高级数字滤波器的设计、仿真与分析工作，提供了基本滤波器体系结构和设计方法，包括自适应滤波器和多速率滤波器，并且可以用于复杂的实时 DSP 系统开发，主要新特性包括：

- 扩展支持二阶 IIR 滤波器，提供滤波器的设计、重构、定标与量化、图形化分析的能力；
- 支持多速率滤波器的设计与分析，将单速率滤波器与多速率滤波器分析设计工作集成；
- 增强的定点滤波器仿真、分析和集成功能，支持 Signal Processing Toolbox；
- FIR 滤波器设计函数；
- 改进 FDATool 工具，包含了高级滤波器的设计方法、多速率滤波器设计功能，改进

了量化设计功能等；
- 支持通过 Filter Design HDL Coder 将滤波器设计结果进行 FPGA 仿真。

4. Instrument Control Toolbox 2

Instrument Control Toolbox 2 提供了与各种仪器设备进行数据通信的能力，该工具箱支持 GPIB、VISA、TCP/IP 以及 UDP 等通信方式。工程师可以在 MATLAB 环境下生成数据发送给相应的仪器设备或者从仪器设备中读取数据并且进行分析和可视化工作，主要的新特性包括：
- 支持 IVI、VXIPlugandPlay 等设备，MATLAB 的仪器驱动无须用户了解仪器特性即可开展工作；
- 新开发的图形界面工具 TMTool，用户管理设备驱动。

5. Mapping Toolbox 2

Mapping Toolbox 2 提供在 MATLAB 环境下进行地理信息显示、分析的函数以及相应的图形化界面工具，主要新特性包括：
- 支持标准 GIS 和地理信息数据文件格式；
- 支持 Transverse Mercator 项目和 PROJ.4 项目数据库。

6. MATLAB Compiler 4

MATLAB Compiler 4 能够将 MATLAB 的算法和应用程序文件转变成可以发布的独立可执行的应用程序， MATLAB Compiler 4 支持更多的 M 语言特性，主要新特性包括：
- 兼容 MATLAB 面向对象数据类型；
- 共享库函数开发环境，兼容 R14 和 R13 版本的程序开发；
- 增强 C++ 语言集成；
- 支持生成各种独立可执行的应用程序，包括 C/C++ 共享库、 COM 组件和 Excel Plug-in 等。

7. MATLAB Report Generator 2

使用 MATLAB Report Generator 2 可以根据用户开发的 MATLAB 应用程序自动创建各种文档报告，主要新特性包括：
- 重新设计的图形界面工具；
- 更快的文档生成速度；
- 支持生成 Adobe PDF 格式文档；
- 增加 MATLAB 组件，支持 Axes、句柄图形以及 MATLAB 属性表格。

8. Optimization Toolbox 3

Optimization Toolbox 3 提供了针对通用问题或者大规模优化问题处理的算法，支持线性规划、二次规划、非线性最小二乘法、非线性方程求解等功能，主要新特性包括：
- 二进制整数规划问题求解；
- 针对中等规模问题实现无约束优化算法函数 fminunc；

➢ 增加使用单纯形算法的线性规划函数 linprog。

9. Signal Processing Blockset 6（原来的 DSP Blockset）

Signal Processing Blockset 6 适用于扩展 Simulink，进行针对帧信号的数字信号处理系统设计、仿真与分析工作。它能够完成通信系统、音频/视频系统、数字控制器、雷达/声呐系统、消费类电子产品以及医疗器械等信号处理系统的开发工作，主要新特性包括：

➢ 音频与语音处理功能，包括 LPC to/from RC，G.711 Codec，CIC 等；
➢ 扩展的数字滤波器，包括 4 类浮点滤波器和 15 类定点结构滤波器；
➢ 增强的定点支持（需要 Simulink Fixed-Point）用于滤波器、信号统计模块等功能；
➢ 新开发的定点参数设置对话框用于设置模型的定点特性，例如字长、二进制小数位以及整数溢出等；
➢ 新改进的 Scope 模块，支持 Waterfall Scope。

10. Virtual Reality Toolbox 4

Virtual Reality Toolbox 4 可以将三维虚拟现实场景引入到 Simulink 模型中，用于 Simulink 模型运算结果的可视化显示工作，主要新特性包括：

➢ 支持动画显示结果的录制；
➢ 支持向量、矩阵数据输入，用于控制场景动画；
➢ 支持从 Simulink 模型中控制动画显示速率；
➢ 改进的三维场景观测器（Viewer），用于模型的创建和观察；
➢ 支持 USB Space Mouse , Space Traveler 运动控制输入设备，以及力回馈操纵杆输入设备。

1.3.2.2 升级产品模块及新模块

MATLAB 7.0 升级的 29 个产品模块和新增的 12 个产品模块列举见表 1-1 至表 1-3。

表 1-1 主要升级模块

模块名称	模块名称
Communications Blockset	Nonlinear Control Design Blockset（更名为 Simulink Response Optimization）
Communications Toolbox	Optimization Toolbox
Control System Toolbox	Real-Time Workshop
Database Toolbox	Real-Time Workshop Embedded Coder
DSP Blockset（更名为 Signal Processing Blockset）	Signal Processing Blockset
Embedded Target for Motorola MPC555	Simulink Fixed Point
Embedded Target for TI C6000 DSP	Simulink Report Generator
Filter Design Toolbox	Simulink Response Optimization
Financial Derivatives Toolbox	Stateflow

续表

模块名称	模块名称
Fixed-Point Blockset （更名为 Simulink Fixed Point）	Stateflow Coder
Instrument Control Toolbox	Statistics Toolbox
Mapping Toolbox	System Identification Toolbox
MATLAB Compiler	Virtual Reality Toolbox
MATLAB Report Generator	Wavelet Toolbox
Model Predictive Control Toolbox	

表 1-2 新产品模块

模块名称	模块名称
Bioinformatics Toolbox	OPC Toolbox
Embedded Target for TI C2000 DSP	RF Blockset
Filter Design *HDL* Coder	RF Toolbox
Fixed-Point Toolbox	Simulink Control Design
Genetic Algorithm and Direct Search Toolbox	Simulink Parameter Estimation
Link for ModelSim	Simulink Verification and Validation

表 1-3 其他升级模块

模块名称	模块名称
Aerospace Blockset 1.6	MATLAB Link for Code Composer Studio 1.3.1
Curve Fitting Toolbox 1.1.1	MATLAB Builder for COM 1.1（formerly named MATLAB COM Builder）
Data Acquisition Toolbox 2.3	MATLAB Builder for Excel 1.2（formerly named MATLAB Excel Builder）
Datafeed Toolbox 1.5	Model-Based Calibration Toolbox 2.1
Dials & Gauges Blockset 1.2	Mu-Analysis and Synthesis Toolbox 3.0.8
Embedded Target for Infineon C166 Microcontrollers 1.1	Neural Network Toolbox 4.0.3
Embedded Target for Motorola HC12 1.1	Partial Differential Equation Toolbox 1.0.5
Embedded Target for OSEK/VDX 1.1	Real-Time Windows Target 2.5
Excel Link 2.2	Robust Control Toolbox 2.0.10
Extended Symbolic Math Toolbox 3.1	Signal Processing Toolbox 6.2
Financial Time Series Toolbox 2.1	SimMechanics 2.2
Financial Toolbox 2.4	SimPowerSystems 3.1
Fixed-Income Toolbox 1.0.1	Simulink Accelerator 6
Fuzzy Logic Toolbox 2.1.3	Symbolic Math Toolbox 3.1

续表

模 块 名 称	模 块 名 称
GARCH Toolbox 2.0.1	xPC Target 2.5
Image Acquisition Toolbox 1.5	xPC Target Embedded Option 2.5
LMI Control Toolbox 1.0.1	xPC TargetBox 2.5

1.4 小　　结

本章介绍了 MATLAB 语言的基础知识，包括 MATLAB 的发展历程、语言特点及最新版本 MATLAB 7.0 的新产品和新功能。通过本章的学习，读者对 MATLAB 可以有一个整体的了解。

第 2 章　MATLAB 程序设计及代码优化

MATLAB 易学易用，其函数名和表达很接近我们书写计算公式的思维表达方式，用 MATLAB 编写程序犹如在演草纸上排列公式与求解问题。MathWorks 公司将 MATLAB 称为第四代编程语言，足见其简捷。同其他的程序设计语言一样，MATLAB 语言提供了流控制、函数、数据结构、输入/输出功能以及面向对象的程序设计方法。

2.1　MATLAB 的表达式和变量

2.1.1　表达式

表达式和变量是 MATLAB 的基础。MATLAB 采用的是表达式语言，用户输入的语句由 MATLAB 系统解释运行。MATLAB 的语句由表达式和变量组成。MATLAB 语句有两种常见的形式：

- 表达式；
- 变量＝表达式。

在第一种形式中，如果不指定将表达式的值赋给某个变量，表达式会将运算产生的结果自动地赋值给名为 ans 的变量，并显示在屏幕上。变量 ans 是一个默认的变量名，它会在以后的类似操作中被自动覆盖。

对于第二种形式，等号右边的表达式计算产生的结果赋值给等号左边的变量，并且放入内存中。

2.1.2　变量

变量是任何程序设计语言的基本要素。与常规的程序设计语言不同，MATLAB 语言的变量不需要事先声明，也不需要指定变量类型，MATLAB 会自动依据所赋予变量的值或对变量所进行的操作来识别变量的类型。赋值过程中如果赋值变量已经存在，将用新值代替旧值，新值类型代替旧值类型。在 MATLAB 中变量的命名应遵循如下规则：

- 变量名区分大小写；
- 变量名长度不超过 31 个字符；
- 变量名以字母开头，可以由字母、数字、下划线组成，但不能使用标点。

MATLAB 语言中的变量也存在变量域的问题。在未加特殊说明的情况下，MATLAB 语言将所识别的一切变量视为局部变量。

2.2　细胞数组与结构数组

2.2.1　细胞数组

用类似矩阵的记号将复杂的数据结构纳入一个变量之下。与矩阵中的圆括号表示下标类

似，细胞数组由大括号表示下标。

```
>> B={1,'Anne cat', 18, [100, 80, 75; 77, 60, 92; 67, 28, 90; 100, 89, 78]}
 B =   [1]     'Anne cat'     [18]     [4x3 double]
```

访问单元数组应该由大括号进行，如第 4 单元中的元素可以由下面的语句得出：

```
>> B{4}
ans = 100   80   75
       77   60   92
       67   28   90
      100   89   78
```

2.2.2 结构数组

MATLAB 的结构体有点像 C 语言的结构体数据结构。每个成员变量用点号表示，如 A.p 表示 A 变量的 p 成员变量。获得该成员比 C 更直观，仍用 A.p 访问，而不用 A->p。用下面的语句可以建立一个小型的数据库。

```
>> student_rec.number=1;
student_rec.name='Alan Shearer';
student_rec.height=180;
student_rec.test=[100, 80, 75; 77, 60, 92; 67, 28, 90; 100, 89, 78];
>> student_rec
student_rec =
number: 1
name: 'Alan Shearer'
height: 180
test: [4x3 double]
```

删除成员变量可以由 rmfield( )函数进行，添加成员变量可以直接用赋值语句即可。另外，数据读取可以由 setfield( )和 getfield( )函数完成。

2.3 类与对象

本节讲解如何创建类来增加 MATLAB 的数据类型。通过创建类和对象，可以增加 MATLAB 的数据类型和操作方法。类定义了一种数据结构以及可用于此种数据的函数和运算符等。对象是类的实例。类与对象是 MATLAB 5.* 开始引入的数据结构。在 MATLAB 手册中定义了一个很好的类——多项式类。事实上，在实际工具箱设计中，用到了很多的类，例如在控制系统工具箱中定义了 LTI（线性时不变系统）类，并在此基础上定义了其子类：传递函数类 TF，状态方程类 SS，零极点类 ZPK 和频率响应类 FR。

下面通过一个例子来介绍类的构造。在 MATLAB 语言使用手册中给出了一个很有代表性的例子：多项式类的建立问题。假设我们想为多项式建立一个单独的类，重新定义加、减、乘及乘方等运算，并定义其显示方式。那么建立一个类至少应该执行下面的步骤：

（1）首先应该选定一个恰当的名字，例如这里的多项式类可选择为 polynom。以这个名字建立一个子目录，目录的名字前加@。对本例来说，即应该在当前的工作目录下建立

@polynom 子目录，而这个目录无须在 MATLAB 路径下再指定。

（2）编写类的构造函数，函数名应该和类同名。在定义类方法的子目录中，必须有一个称为构造程序的 M 文件。构造程序通过初始化类的数据结构和给类分配标号来定义类。

下面是多项式类的构造函数 polynom(a)的内容：

```
function p = polynom(a)
%POLYNOM Polynomial class constructor.
%    p = POLYNOM(v) creates a polynomial object from the vector v,
%    containing the coefficients of descending powers of x.
if nargin == 0
    p.c = [];
    p = class(p,'polynom');
elseif isa(a,'polynom')
    p = a;
else
    p.c = a(:).';
    p = class(p,'polynom');
end
```

可以看出，当采用无输入参数的格式调用函数 polynom( )时，构造程序将为该对象生成一个模板，通常情况下属性值为空；当输入参数为多项式时，构造程序将原封不动地返回该多项式；当输入参数不是多项式时，构造程序将把输入参数变形为行向量并赋给属性 c，最后用函数 class( )加上标号“polynom”，表明该对象为多项式。

一般来说，类的构造函数都采用类似 polynom(a)的框架，大致包含以下几个部分：

- 如果不给输入变量，则建立一个空的多项式；
- 如果输入变量 a 已经为多项式类，则将它直接传送给输出变量 p；
- 如果 a 为向量，则将此向量变换成行向量，再构造成一个多项式对象；
- 该结构的属性赋值；
- 用函数 class( )加标号。

（3）如果想正确地显示新定义的类，则必需首先定义 display( ) 函数，并对新定义的类重新定义其基本运算。对多项式来说，可以如下定义有关的函数：要改变显示函数的定义，则需在此目录下重新建立一个新函数 display( )。这种重新定义函数的方法又称为函数的重载。显示函数可以如下地重载定义：

```
function display(p)
% POLYNOM/DISPLAY Command window display of a polynom
disp(' ');
disp([inputname(1),' = '])
disp(' ');
disp(['     ' char(p)])
disp(' ');
```

从上面的定义可见，显示函数要求重载定义 char( ) 函数，用于把多项式转换成可显示的字符串。该函数的定义为：

```
function s = char(p)
% POLYNOM/CHAR
% CHAR(p) is the string representation of p.c
if all(p.c == 0)
    s = '0';
else
    d = length(p.c) - 1;
    s = [];
    for a = p.c;
        if a ~= 0;
            if ~isempty(s)
                if a > 0
                    s = [s ' + '];
                else
                    s = [s ' - '];
                    a = -a;
                end
            end
            if a ~= 1 | d == 0
                s = [s num2str(a)];
                if d > 0
                    s = [s '*'];
                end
            end
            if d >= 2
                s = [s 'x^' int2str(d)];
            elseif d == 1
                s = [s 'x'];
            end
        end
        d = d - 1;
    end
end
```

仔细研究此函数，可以发现，该函数能自动地按照多项式显示的格式构造字符串。比如，多项式各项用加减号连接，系数与算子之间用乘号连接，而算子的指数由 ^ 表示。再配以显示函数，则可以将此多项式以字符串的形式显示出来。

➢ 双精度处理

双精度转换函数的重载定义是很简单的。

```
function c = double(p)
```

```
% POLYNOM/DOUBLE   Convert polynom object to coefficient vector.
%     c = DOUBLE(p) converts a polynomial object to the vector c
%     containing the coefficients of descending powers of x.
c = p.c;
```

➢ 加运算

两个多项式相加，只需将其对应项系数相加即可。这样，加法运算的重载定义可由下面的函数实现。注意，这里要对 plus( ) 函数进行重载定义。

```
function r = plus(p,q)
% POLYNOM/PLUS   Implement p + q for polynoms.
p = polynom(p);
q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] + [zeros(1,-k) q.c]);
```

同理，还可以重载定义多项式的减法运算：

```
function r = minus(p,q)
% POLYNOM/MINUS Implement p - q for polynoms.
p = polynom(p);
q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] - [zeros(1,-k) q.c]);
```

➢ 乘法运算

多项式的乘法实际上可以表示为系数向量的卷积，可以由 conv( )函数直接获得，故可以如下重载定义多项式的乘法运算。

```
function r = mtimes(p,q)
% POLYNOM/MTIMES    Implement p * q for polynoms.
p = polynom(p);
q = polynom(q);
r = polynom(conv(p.c,q.c));
```

➢ 乘方运算

多项式的乘方运算只限于正整数乘方的运算，其 n 次方相当于将该多项式自乘 n 次。若 n=0，则结果为 1。这样我们就可以重载定义多项式的乘方运算为：

```
function p=mpower(a,n)
if n>=0, n=floor(n); a=polynom(a); p=1;
if n>=1,
for i=1:n, p=p*a; end
end
else, error('Power should be a non-negative integer.')
end
```

（4）定义了类之后，我们就可以方便地进行多项式处理了。例如我们可以建立两个多项

```
    case b                      %得分在 80 和 89 之间
        S(i).Rank=' 良好';       %列为'良好'等级
    case c                      %得分在 60 和 79 之间
        S(i).Rank=' 及格';       %列为'及格'等级
    otherwise                   %得分低于 60
        S(i).Rank='不及格';      %列为'不及格'等级
    end
end
%将学生姓名，得分，登记等信息打印出来
disp(['学生姓名 ',' 得分 ',' 等级']);disp(' ')
for i=1:5;
 disp([S(i).Name,blanks(6),num2str(S(i).Marks),blanks(6),S(i).Rank]);
end
```

程序运行结果如下：

```
学生姓名    得分      等级
 Jack       72        及格
 Marry      83        良好
 Peter      56        不及格
 Rose       94        优秀
 Tom        100       满分
```

2.4.5 try-catch 结构

【例 2-6】try-catch 结构应用实例。

```
N=4;A=magic(3);            %设置 3 行 3 列矩阵 A
try
    A_N=A(N,:),            %取 A 的第 N 行元素
catch
      A_end=A(end,:),      %如果取 A(N,:)出错，则改取 A 的最后一行
end
lasterr                    %显示出错原因
```

程序运行结果如下：

```
A_end =
      4     9     2
ans =
 Index exceeds matrix dimensions
```

2.5 M 文件编程

计算机程序就是计算机指令的集合，不同编程语言的指令与功能是不一样的。MATLAB 语言是一种面向对象的高级语言，它具有编程效率高、易学易用的优点。本节不对 MATLAB

的编程语法进行详细介绍，而是介绍一些与 MATLAB 混合编程有关的基础知识。MATLAB 有两种常用工作方式：一种是直接交互的指令行操作方式，在这种方式下，MATLAB 被当做一种高级“数学演算和图形显示器”来使用。另一种是 M 文件编程方式。

M 文件有两种类型：脚本文件和函数文件。它们的扩展名相同，都是“.m”。脚本文件是一种简单的 M 文件，如同 DOS 下的批处理文件，由一连串的 MATLAB 命令组成，执行时依次执行。脚本文件中的语句可以访问 MATLAB 工作区（Workspace）中的所有数据，运行过程中产生的所有变量都是全局变量。函数文件则接收输入的参数，然后执行并输出结果。下面是一个标准的函数结构：

```
function [y1,y2]=functionname(x1,x2,...xn)        //函数定义行
%                                                  //函数注释部分
% Parameters:   y1,y2:                             //输出参数
%               x1, x2,...xn:                      //输入参数
%
MATLAB 语句                                        //函数主体
end
```

在上面的完整函数结构中，function 是保留字。y1,y2 是输出参数，x1,x2,...,xn 是输入参数。以%开头的每一行是函数的注释部分，主要用于注释一些函数帮助信息，如函数的作用，输入/输出参数以及版本信息等。在 MATLAB 环境下，通过 help 命令得到的函数帮助信息都是在这一部分定义的。

其实，脚本文件与函数文件的区别在于脚本文件没有函数定义行，且一般没有注释信息，也没有诸如 begin/end 之类的分界符。当然，脚本文件也可以添加注释信息，以%开头的都是注释信息。注释信息既可以单独成行，也可以放在 MATLAB 语句的后面。相应地，脚本文件与函数文件在使用方法、变量生存周期上也存在着差异，归纳见表 2-1。值得说明的是，尽管脚本文件没有返回值参数，但脚本文件中的变量在脚本文件执行后仍保留在工作区间。

表 2-1　脚本文件和函数文件的比较

	脚 本 文 件	函 数 文 件
输入/输出参数	没有输入/输出参数，不能接收和输出数据	带有输入和输出参数，可以接收和输出数据
变量生存周期	操作工作区间中的数据	默认内部变量为函数局部变量，工作区间不能访问
适用场合	对于需要多次执行的一系列命令特别有用	需要多次执行，而且需要带输入/输出参数

因此，脚本文件和函数文件适合于不同的场合。在后面的章节中将会看到，许多情况下需要函数文件，因此需要将脚本文件转换为函数文件。转换方法实际上非常简单，只需要在脚本文件前面添加必要的函数定义行和注释信息（可以省略）即可。下面以一个简单的脚本文件为例进行说明。

假设有一个名称为 lin_intp.m 的脚本文件。该函数的功能是假设一条直线通过（x1,y1）和（x2,y2）两点，要求任一点 x 处的 y 值。脚本文件的内容如下：

```
%An Script M-file to find a new value using linear interpolation
x1=1.0;
x2=1.50;
y1=29.3;
y2=33.7;
x=1.10;
newvalue=y1+((x-x1)/(x2-x1))*(y2-y1)
```

在 MATLAB 环境下，输入 lin_intp，得到 newvalue=30.1800，即

```
>> lin_intp
newvalue =
    30.1800
```

现在要将脚本文件转换为函数文件。如果不需要计算其他 x 点处的 y 值，可以简单地改为（设函数文件名为 l_interp.m）：

```
function newvalue=l_interp( )
x1=1.0;
x2=1.50;
y1=29.3;
y2=33.7;
x=1.10;
newvalue=y1+((y2-y1)/(x2-x1))*(x-x1);
```

在 MATLAB 环境下，输入 l_interp 得到的返回值为 30.1800，即

```
>> l_interp
ans =
    30.1800
```

函数 l_interp 只能得到点 x=1.10 处的 y 值，可以通过将 x 改为输入参数以增加灵活性。假设新函数文件名为 l_interpx.m，则函数 l_interp 可改写为：

```
function newvalue=l_interpx(x)
x1=1.0;
x2=1.50;
y1=29.3;
y2=33.7;
newvalue=y1+((y2-y1)/(x2-x1))*(x-x1);
```

要计算点 x=1.10 处的 y 值，可以在 MATLAB 环境下键入 l_interpx(1.10)即

```
>> l_interpx(1.10)
ans =
    30.1800
```

函数 l_interpx( )是在直线通过两点（x1, y1）和（x2, y2）都已知的情况下的函数。为进一步增强通用性，即已知过任意两点，求另一点 x 对应的 y 值。假设新函数名为 l_intpolation.m,

则函数 l_interpolation 可改写为：

```
function newvalue=l_interpolation(x1,y1,x2,y2,x)
% parameters:
% Input:
%       x1,y1:  the location of the first pixel
%       x2,y2:  the location of the second pixel
%       x     :  the x location of the third pixel
% Output:
%       newvalue: the y location of the third pixel to be calculated
newvalue=y1+((y2-y1)/(x2-x1))*(x-x1);
```

若要计算通过两点（1.0, 29.3）和（1.5, 33.7）的直线在点 x=1.10 处的 y 值，则在 MATLAB 环境中键入 l_interpolation（1.0, 29.3, 1.5, 33.7, 1.10），即

```
>> l_interpolation(1.0, 29.3, 1.5, 33.7, 1.10)
ans =
   30.1800
```

函数 l_interpolation( )增加了参数说明，是因为它的输入参数较多，必须进行说明以方便用户使用。

通过本例可以看出，脚本文件转换为函数文件并不是唯一的，可以根据问题对函数灵活性的要求进行适当的转换。

2.6 M 文件编程规范

本编程规范主要着眼于提高 M 文件的正确性、规范性和通用性，而不是在于提高 M 程序的效率。当然，编程规范并不是强制性的。M 文件规范大致与 C、C++和 Java 的编程规范相同，有些针对 MATLAB 的特点进行了修改，接下来突出介绍几个这方面的规范。

（1）函数名和文件名必须相同。例如，函数 fliplr 存储在名为 fliplr.m 的文件中。

（2）变量的名字应该能够反映它们的意义或者用途。小范围应用的变量应该用短的变量名，实际上，大多数变量都应该是有意义的变量名，短变量名通常作为结构申明时必须阐明变量的保留用法。当然，作为“草稿变量”的临时存储空间或者索引可以用短名字。

（3）结构体的命名应该以一个大写字母开头，这与 C++实际编程规范一致，有助于区分结构体与普通变量。结构体的命名应该是暗示性的，并且不需要包括字段名。例如 Segment.length 相对 Segment.SegmentLength 来说更加合理。

（4）MATLAB 第一次执行一个 M 文件函数时，将打开相应的文本文件并将命令编辑成存储器的内部表示，以加速执行以后所有的调用。如果函数包含了对其他 M 文件函数的调用，它们也同样被编译到存储器中。普通的脚本 M 文件不被编译，即使它们是从函数 M 文件内调用；每次打开脚本 M 文件时，都逐行进行注释。

（5）MATLAB 采用结构化的程序设计。编写一个大型的程序时，最好将它划分为一些小的模块，通常采用函数的方式，以增强程序的可读性和可测试性。

（6）MATLAB 支持函数间的相互调用。

M 文件可以包含两个以上的函数，第一个函数就是主函数，通过 M 文件的名字就可以调用这个函数。除主函数外，其余的函数就是子函数。子函数只能被本文件中的主函数和子函数调用。每一个子函数以它自己的函数定义行开始。主函数必须排在前面，子函数的顺序可以任意排列。下面是一个利用子函数的例子：

```
function [avg, med]=newstats(u)
% newstatus: 利用内部函数计算平均值和中间数
n=length(u);
avg=mean(u, n);
med=median(u, n);
function a=mean(v, n)        //子函数
% mean: 计算平均值
a=sum(v)/n;
function mid=median(v, n)    //子函数
% median: 计算中间数
w=sort(v);
if rem(n, 2)==1
    m=w((n+1)/2);
else
    m=(w(n.2)+w(n/2+1))/2;
end
```

（7）自己编写的函数最好在编写时就添加注释，包括输入/输出参数、解释用法以及可能需要的改进等信息，以增强函数的可读性和可用性。经验表明，在写代码的同时就加上注释比后来再补充注释要好。

（8）函数可以没有或具有一个或多个输入参数，也可以没有或具有一个或多个输出参数。

（9）函数可以按少于函数 M 文件中所规定的输入和输出变量进行调用，但不能多于函数 M 文件中所规定的输入和输出变量数目。如果输入和输出变量数目多于函数 M 文件中 function 语句所规定的数目，则调用时自动返回一个错误。

（10）当函数有一个以上输出变量时，输出变量包含在括号内。例如，[V, D] = eig(A)。

（11）当调用一个函数时，输入和输出参数的数目在函数内是规定好的。函数工作空间变量 nargin 包含输入参数的个数；函数工作空间变量 nargout 包含输出参数的个数。事实上，这些变量常用来设置默认输入变量，并决定用户所希望的输出变量。例如，MATLAB 提供的函数 linspace( )如下：

```
function y = linspace(d1, d2, n)
%LINSPACE Linearly spaced vector.
%    LINSPACE(X1, X2) generates a row vector of 100 linearly
%    equally spaced points between X1 and X2.
%
%    LINSPACE(X1, X2, N) generates N points between X1 and X2.
%    For N < 2, LINSPACE returns X2.
```

```
%
%    See also LOGSPACE, :.

%    Copyright 1984-2002 The MathWorks, Inc.
%    $Revision: 5.12 $  $Date: 2002/02/05 13:47:28 $

if nargin == 2
    n = 100;
end

y = [d1+(0:n-2)*(d2-d1)/(floor(n)-1) d2];
```

这里，如果用户只用两个输入参数调用 linspace( )，例如 linspace(0,10)，则产生 100 个数据点。相反，如果输入参数的个数是 3，例如 linspace(0,10,50)，则第三个参数决定数据点的个数，即只产生 50 个数据点。

（12）函数可以有一个或多个输出参数。

例如 MATLAB 提供的函数 size( )。这个函数不是一个 M 文件函数（它是一个内置函数），它的帮助文本说明了输出参数的选择。

```
function [varargout] = size(varargin)
%SIZE    Size of array.
%    D = SIZE(X), for M-by-N matrix X, returns the two-element
%    row vector D = [M, N] containing the number of rows and columns
%    in the matrix.  For N-D arrays, SIZE(X) returns a 1-by-N
%    vector of dimension lengths.  Trailing singleton dimensions
%    are ignored.
%    [M,N] = SIZE(X) for matrix X, returns the number of rows and
%    columns in X as separate output variables.
%    [M1,M2,M3,...,MN] = SIZE(X) returns the sizes of the first N
%    dimensions of array X.  If the number of output arguments N does
%    not equal NDIMS(X), then for:
%
%    N > NDIMS(X), size returns ones in the "extra" variables,
%                  i.e., outputs NDIMS(X)+1 through N.
%    N < NDIMS(X), MN contains the product of the sizes of the
%                  remaining dimensions, i.e., dimensions N+1 through
%                  NDIMS(X).
%
%    M = SIZE(X,DIM) returns the length of the dimension specified
%    by the scalar DIM.  For example, SIZE(X,1) returns the number
%    of rows.
%
%    When SIZE is applied to a Java array, the number of rows
```

```
%    returned is the length of the Java array and the number of columns
%    is always 1.   When SIZE is applied to a Java array of arrays, the
%    result describes only the top level array in the array of arrays.
%
%    See also LENGTH, NDIMS, NUMEL.

%    Copyright 1984-2004 The MathWorks, Inc.
%    $Revision: 5.13.4.3 $   $Date: 2004/04/16 22:06:29 $
%    Built-in function.

if nargout == 0
  builtin('size', varargin{:});
else
  [varargout{1:nargout}] = builtin('size', varargin{:});
end
```

如果函数仅出现一个输出参数，则返回一个二元素的行，包含行数和列数。相反，如果出现两个输出参数，则 size( )分别返回行和列。在 M 文件函数里，变量 nargout 可用来检验输出参数的个数，并按要求修正输出变量的创建。

（13）一个函数说明一个或多个输出变量，但没有要求输出时，就简单地不给输出变量赋任何值。

例如，MATLAB 提供的计时结束函数 toc( )可以说明这个属性。

```
function t = toc
%TOC Read the stopwatch timer.
%    TOC, by itself, prints the elapsed time (in seconds) since TIC was used.
%    t = TOC; saves the elapsed time in t, instead of printing it out.
%
%    See also TIC, ETIME, CLOCK, CPUTIME.

%    Copyright 1984-2003 The MathWorks, Inc.
%    $Revision: 5.12.4.2 $   $Date: 2004/04/10 23:33:08 $

% Note that toc is implemented as a builtin function for efficiency. Any changes
% to this file will not change the behavior of toc. This file is provided as
% an example of how to use the 'global' keyword only.

% TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if isempty(TICTOC)
  error('MATLAB:toc:callTicFirst', 'You must call TIC before calling TOC.');
end
if nargout < 1
```

```
    disp(sprintf('Elapsed time is %f seconds.', etime(clock,TICTOC)));
else
    t = etime(clock,TICTOC);
end
```

如果用户不以输出参数调用 toc( )，例如在 MATLAB 环境下直接输入“»toc”，则不指定输出变量的值，函数在命令窗口显示函数工作空间变量 elapsed_time，但在 MATLAB 工作空间里不创建变量。相反，如果 toc( )是以“» out=toc”调用，则按变量 out 将消逝的时间返回到命令窗口。

（14）函数有它们自己的专用工作空间，它与 MATLAB 的工作空间分开。

函数内变量与 MATLAB 工作空间之间唯一的联系是函数的输入和输出变量。如果函数的任意一个输入变量值发生变化，则其变化仅在函数内出现，不影响 MATLAB 工作空间的变量。函数内所创建的变量只驻留在函数的工作空间，而且只在函数执行期间临时存在，以后就消失。因此，从一个调用到下一个调用，在函数工作空间存储信息是不可能的。然而使用全局变量就提供这个能力。

（15）如果一个预定的变量在 MATLAB 工作空间重新定义，则不会延伸到函数的工作空间。逆向有同样的属性，即函数内的重新定义变量不会延伸到 MATLAB 的工作空间中。

（16）当调用一个函数时，输入变量不会拷贝到函数的工作空间，但使它们的值在函数内可读。然而改变输入变量内的任何值，数组就拷贝到函数工作空间。如果输出变量与输入变量相同，例如，函数 x=fun（x, y, z）中的 x，那么就将它拷贝到函数的工作空间。因此，为了节约存储和加快速度，最好从大数组中抽取元素，然后对它们进行修正，而不是使整个数组拷贝到函数的工作空间。

（17）如果变量说明是全局的，函数可以与其他函数、MATLAB 工作空间和递归调用本身共享变量。为了在函数内或 MATLAB 工作空间中访问全局变量，在每一个所希望的工作空间，变量必须说明是全局的。

全局变量的使用以 MATLAB 提供的函数 tic 和 toc 为例，它们合在一起工作。

```
function tic
%TIC Start a stopwatch timer.
%    The sequence of commands
%        TIC, operation, TOC
%    prints the number of seconds required for the operation.
%
%    See also TOC, CLOCK, ETIME, CPUTIME.

%    Copyright 1984-2002 The MathWorks, Inc.
%    $Revision: 5.11.4.1 $  $Date: 2003/10/16 04:52:00 $

% Note that tic is implemented as a builtin function for efficiency. Any changes
% to this file will not change the behavior of tic. This file is provided as
% an example of how to use the 'global' keyword only.
```

```
% TIC simply stores CLOCK in a global variable.
global TICTOC
TICTOC = clock;

function t = toc
%TOC Read the stopwatch timer.
%     TOC, by itself, prints the elapsed time (in seconds) since TIC was used.
%     t = TOC; saves the elapsed time in t, instead of printing it out.
%
%     See also TIC, ETIME, CLOCK, CPUTIME.

%     Copyright 1984-2003 The MathWorks, Inc.
%     $Revision: 5.12.4.2 $   $Date: 2004/04/10 23:33:08 $

% Note that toc is implemented as a builtin function for efficiency. Any changes
% to this file will not change the behavior of toc. This file is provided as
% an example of how to use the 'global' keyword only.

% TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if isempty(TICTOC)
   error('MATLAB:toc:callTicFirst', 'You must call TIC before calling TOC.');
end
if nargout < 1
    disp(sprintf('Elapsed time is %f seconds.', etime(clock,TICTOC)));
else
    t = etime(clock,TICTOC);
end
```

在函数 tic( )中，变量 TICTOC 说明为全局的，因此它的值由调用函数 clock( )来设定。以后在函数 toc( )中，变量 TICTOC 也说明为全局的，让 toc( )访问存储在 TICTOC 中的值。利用这个值，toc( )计算自执行函数 tic( )以来消逝的时间。值得注意的是，变量 TICTOC 存在于 tic( )和 toc( )的工作空间中，而不在 MATLAB 工作空间中。

（18）实际编程中，无论什么时候应尽量避免使用全局变量。

如果用了全局变量，建议全局变量名要长，可包含所有的大写字母，并有选择地以首次出现的 M 文件的名字开头。如果遵循建议，则可以将全局变量之间不必要的互作用减至最小。例如，如果另一函数或 MATLAB 工作空间说明 TICTOC 为全局的，那么它的值在该函数或 MATLAB 工作空间内可被改变，而函数 toc 会得到不同的、可能是无意义的结果。

（19）若在 MATLAB 中输入一个符号，则按变量、内置函数和外部函数的顺序搜索。例如，输入“»cow”，则 MATLAB 首先认为 cow 是一个变量。如果它不是，那么 MATLAB 认为它是一个内置函数。如果还不是，MATLAB 则在 PATH 所规定的路径上检查外部函数，如果没有找到则提示出错。

（20）从函数 M 文件内可以调用脚本文件。在这种情况下，脚本文件查看函数工作空间，不查看 MATLAB 工作空间。从函数 M 文件内调用的脚本文件不必用调用函数编译到内存。函数每调用一次，它们就被打开和解释。因此，从函数 M 文件内调用脚本文件减慢了函数的执行。

（21）函数可以递归调用，即 M 文件函数能调用它们本身。例如，考虑一个函数 iforgot( )：

```
function iforgot(n)
%   IFORGOT Recursive Function Call Example
%   Copyright (c) 1996 by Prentice-Hall,Inc

if nargin==0,n=20;end
if n>1
    disp(' I will remember to do my homework. ')
    iforgot(n-1)
else
    disp(' Maybe NOT! ')
end
```

调用这个函数出现如下情况：

```
» iforgot(10)
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 I will remember to do my homework.
 Maybe NOT!
```

递归调用函数功能在许多应用场合是有用的。在编制要递归调用的函数时，必须确保会终止，否则 MATLAB 会陷入死循环。最后，在一个递归函数内，如果变量说明是全局的，则该全局变量对以后所有函数调用是可用的。在这个意义下，全局变量变成静态的，并在函数调用之间不会消失。

（22）当函数 M 文件到达 M 文件终点，或者碰到返回命令 return 时，结束执行和返回。return 命令提供了一种结束一个函数的简单方法，而不必到达文件的终点。

（23）MATLAB 函数 error( )在命令窗口显示一个字符串，放弃函数执行，把控制权返回给键盘。这个函数对提示函数使用不当很有用，例如在以下文件片段中：

```
if length(val)>1
  error(' VAL must be a scalar. ')
end
```

这里，如果变量 val 不是一个标量，则 error( )显示消息字符串，把控制权返回给命令窗口和键盘。

（24）当一个函数的输入参数的个数超出了规定的范围时，MATLAB 函数 nargchk( )提供了统一的响应。函数 nargchk( )为：

```
function [varargout] = nargchk(varargin)
%NARGCHK Validate number of input arguments.
%    MSGSTRUCT = NARGCHK(LOW,HIGH,N,'struct') returns an appropriate error
%    message structure if N is not between LOW and HIGH. If N is in the
%    specified range, the message structure is empty. The message structure
%    has at a minimum two fields, 'message' and 'identifier'.
%
%    MSG = NARGCHK(LOW,HIGH,N) returns an appropriate error message string if
%    N is not between LOW and HIGH. If it is, NARGCHK returns an empty matrix.
%
%    MSG = NARGCHK(LOW,HIGH,N,'string') is the same as
%    MSG = NARGCHK(LOW,HIGH,N).
%
%    Example
%       error(nargchk(1, 3, nargin, 'struct'))
%
%    See also NARGOUTCHK, NARGIN, NARGOUT, INPUTNAME, ERROR.

%    Copyright 1984-2002 The MathWorks, Inc.
%    $Revision: 5.12.4.3 $  $Date: 2003/11/06 15:42:09 $
%    Built-in function.

if nargout == 0
  builtin('nargchk', varargin{:});
else
  [varargout{1:nargout}] = builtin('nargchk', varargin{:});
end
```

可以通过"error(nargchk(nargin, 2, 5))"说明在一个 M 文件函数内的典型用法。如果 nargin 的值小于 2，函数 error( )像前面描述的那样进行处理，nargchk( )返回字符串"没有足够的输入参数"如果 nargin 的值大于 5，函数 error( )执行处理，nargchk( )返回字符串"太多输入参数"如果 nargin 在 2～5 之间，函数 error( )简单地将控制传递给下一个语句，nargchk( )返回一个空字符串。也就是说，当它的输入参数为空时，error( )函数什么也不做。

（25）MATLAB 运行时缓存了 Toolbox 子目录及其子目录中所有 M 文件的名字和位置，以便 MATLAB 可以很快地找到和执行函数 M 文件。被缓存的 M 文件被当做是只读的。如果执行这些函数，以后发生变化 MATLAB 将只执行以前编译到内存的函数，不管已改变的 M 文件。而且在 MATLAB 执行后，如果 M 文件被加到 Toolbox 目录中，那么它们将不出现在缓存里，因此不可利用。所以，在 M 文件函数的使用中，最好把它们存储在 Toolbox 目录外，

或存储在 MATLAB 目录下，直至它们被认为是完备的。当它们完备时，将它们移到一个只读的 Toolbox 目录或其子目录内。

（26）在 Toolbox 目录外，MATLAB 跟踪 M 文件的修改日期。所以，当遇到一个以前编译到内存的 M 文件函数时，MATLAB 把已编译的 M 文件的修改日期与在磁盘上的 M 文件比较。如果日期是相同的，MATLAB 执行已编译的 M 文件。相反，如果在磁盘上的 M 文件是新的，MATLAB 清除以前已编译的 M 文件，且编译这个新的和修改过的 M 文件。

（27）在 mfilename( )函数内，有要执行的 M 文件的名字。例如，正在执行 M 文件 function.m 时，函数的工作空间包含变量 mfilename，它包含函数字符串。这个变量也存在于脚本文件里，在这种情况下，它包含了要执行的脚本文件的名字。

总之，函数 M 文件提供了一个简单的扩展 MATLAB 功能的方法。事实上，MATLAB 本身的许多标准函数就是 M 文件函数。

2.7 M 文件评述器

M 文件具有方便的矩阵和数组运算，编程效率高，但是运算效率较低，即与传统的 Fortran、C 相比，实现相同功能的 M 代码更简捷、高效，但运算时间较长。一般而言，都希望程序运行速度快一些。在自己编写 MATLAB 程序代码时，如果执行效率不满意，往往希望找到程序执行缓慢的瓶颈。为此，MATLAB 提供了一个称为评述器（Profile）的优化和调试工具，帮助用户在程序中查找哪些代码最需要运算时间。

评述文件是通过命令 profile 来实现的，其调用方式为：profile 关键字。其中，关键字可为：on、off、resume、clear、viewer 等。

下面通过一个具体的例子来说明如何使用 profile 分析 M 文件的瓶颈。MATLAB 环境下输入：

```
profile on
plot(magic(35))
profile viewer
profsave(profile('info'),'profile_results')
profile on -history
plot(magic(4));
p = profile('info');
for n = 1:size(p.FunctionHistory,2)
    if p.FunctionHistory(1,n)==0
        str = 'entering function: ';
    else
        str = ' exiting function: ';
    end
    disp([str p.FunctionTable(p.FunctionHistory(2,n)).FunctionName]);
end
```

运行结果如图 2-1 所示。注意，得到的统计数据，如 Total recorded time 等，会因为所使用的计算机性能的不同而不同。从图中可以看到概略的统计结果，点击每个函数后可以看到更详细的统计结果，如图 2-2～图 2-4 所示。

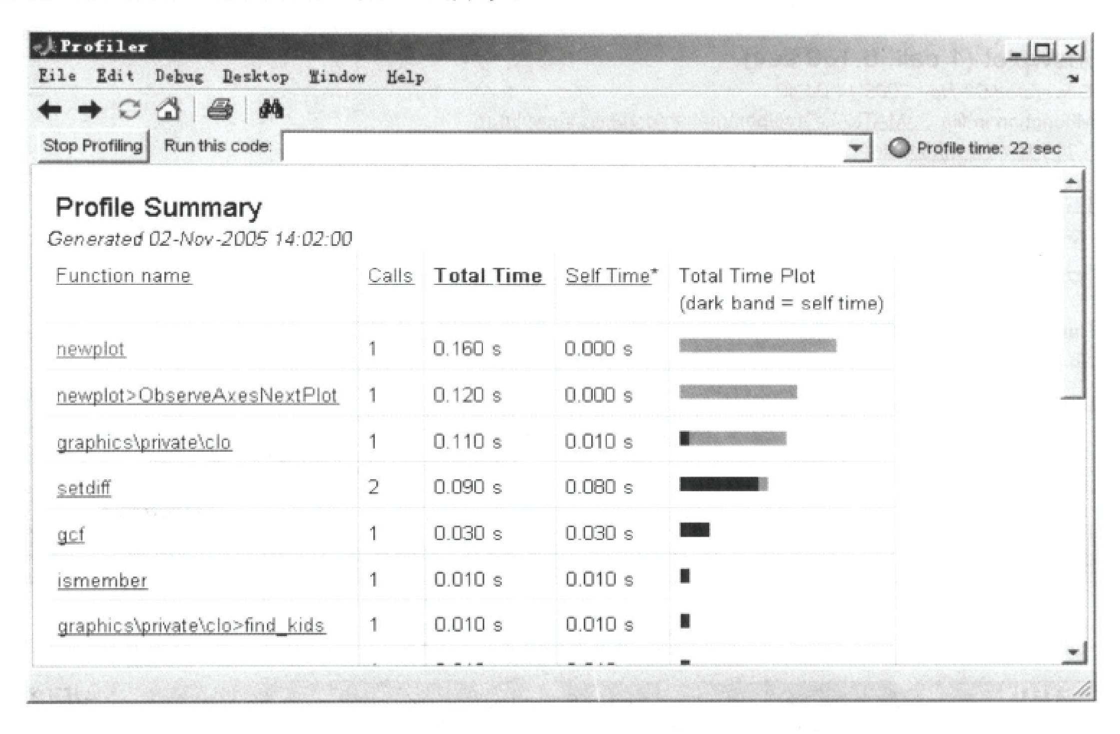


图 2-1　文件评述器运行界面

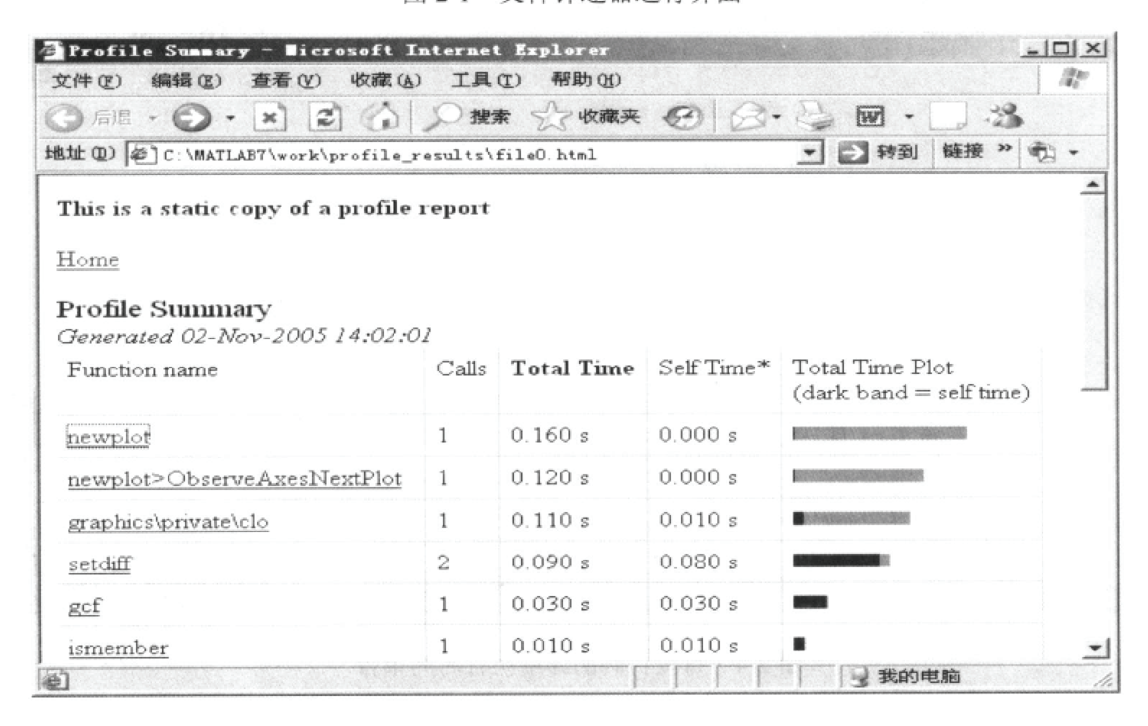


图 2-2　生成的超文本报告



图 2-3 每个函数的详细报告

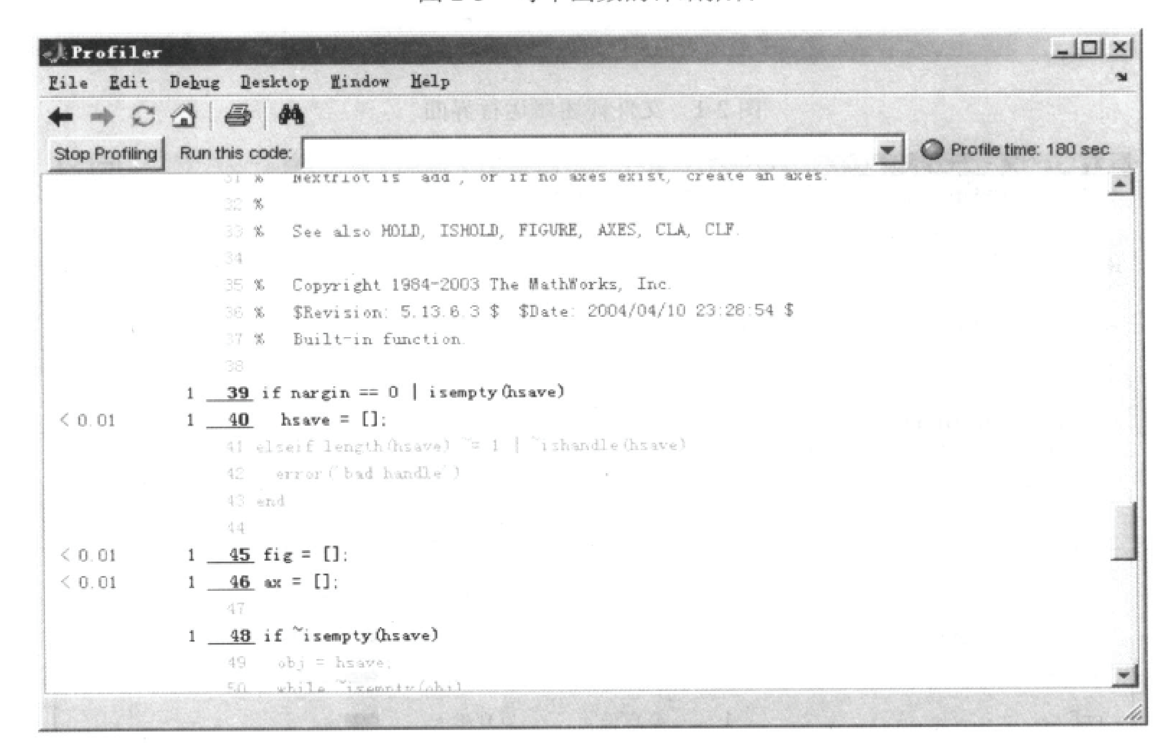


图 2-4 每段代码的详细执行时间报告

2.8 提高 M 文件执行效率的技巧

MATLAB 是一种解释语言，执行效率总体上比 C、Fortran 等通用编程语言低。因此，编程时需要尽可能地提高 M 文件的执行效率。一方面可以利用 MATLAB 述评器分析编写的代码中影响 M 文件效率的瓶颈所在，然后运用一些编程技巧改进程序的执行效率。下面归纳几

种提高 M 文件执行效率的技巧。

2.8.1 矢量化操作

MATLAB 变量的基本类型是矩阵，因为它主要是为向量和矩阵操作设计的。当对矩阵的每个元素循环处理时，运算速度很慢。因此，编程时应尽量对矢量和矩阵整体编程，而不是如同其他的程序设计语言，循环操作矩阵的元素。可以采用矢量化操作，把循环矢量化，这样既可以提高编程效率，也可以提高程序的执行效率。

下面是一段常见的循环程序段：

```
i = 0;
for t = 0:.01:100
    i = i+1;
    y(i) = sin(t);
end
```

可以采用矢量化操作，改写为：

```
t = 0:.01:100;
y = sin(t);
```

MATLAB 提供了专门的函数以测试程序的运行时间。函数 tic 为计时开始，toc 为计时结束。在 MATLAB 环境下运行上面的循环程序：

```
>> tic;
i = 0;
for t = 0:.01:100
    i = i+1;
    y(i) = sin(t);
end
toc;

elapsed_time =
   12.3680
```

上面的短短一段程序需要耗时 12.368s。在 MATLAB 环境下运行矢量化后的程序段：

```
>> tic;
t = 0:.01:100;
y = sin(t);
toc;

elapsed_time =
0.0400
```

上面的程序段只需要 0.04s，可见通过矢量化操作可使执行效率提高约 300 倍。

注：本书列举的执行时间是在 Windows 2000 平台、Pentium 700M CPU 上取得的，仅供参考。当你在相同的条件下执行相同的代码时，执行时间也可能略有差异。本书后面的例子与此相同。

MATLAB 提供的常用于矢量化操作的函数有：all、any、diff、permute、repmat、logical、find、sort 和 sum 等。例如，语句 repmat（A，m,n）将输入矩阵 A 横向和纵向重复。使用函数 repmat( )的例子为：

```
>> A = [1 2 3; 4 5 6];
>> B = repmat(A, 2, 3);
>> B

B =
     1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6
     1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6
```

2.8.2 给数组预定义维

MATLAB 中变量在使用之前不需要明确地定义和指定维数。但当未预定义数组或矩阵的维数时，每当需赋值的元素下标超出现有的维数时，MATLAB 就为该数组或矩阵扩维一次，这会大大降低程序的执行效率。因此，数组预定义维可以提高程序的执行效率。此外，给数组预定义维可以提高内存的使用效率，否则数组的多次扩充维会增加内存的碎片。

将第 2.8.1 节中的循环程序改写如下：

```
tic;
i = 0;
y=size(1,10000);      %为变量 y 预定义维
for t = 0:.01:100
    i = i+1;
    y(i) = sin(t);
end
toc;
```

在 MATLAB 环境下运行，则：

```
elapsed_time =
    5.5580
```

可见，通过给变量 y 预定义维，执行时间从 12.3680s 缩短为 5.5580s，效率提高了。

MATLAB 提供的给数组预定义维的函数主要有 zeros，cell 等，它们的用法概括于表 2-2 中。

表 2-2 给数组预定义维的函数

数组类型	函数	用法
数学数组	zeros	y = zeros(1,100);
细胞数组	cell	B = cell(2,3); B{1,3} = 1:3; B{2,2} = 'string';
矩阵/数组	min	min(A), A 为其矩阵，返回矩阵中最小的元素
矩阵/数组	max	max(A), A 为其矩阵，返回矩阵中最大的元素
结构数组	struct repmat	data = repmat(struct('x',[1 3],... 'y',[5 6]), 1, 3);

2.8.3 下标或者索引操作

在 MATLAB 中，矩阵元素的引用可用两个下标来表示。例如，A(i, j)表示矩阵第 i 行第 j 列的元素；A(1:k, j)表示矩阵 A 的第 j 列的前 k 个元素；A(:, j)表示矩阵的第 j 列的所有元素。求矩阵 A 的第 j 列元素之和的表达式为 sum(A(:, j))。

在图像处理中，特别是视频压缩编码中，都是按宏块或子块进行编码的。例如，假设 x 为原始图像，y 为编码后的图像，可以采用下面的程序段计算编码后图像的峰值信噪比 PSNR：

```
d = mean( mean( (x-y).^2 ) );
m = max( max(x(:)),max(y(:)) );
PSNR = 10*log10( m/d );
```

2.8.4 尽量多使用函数文件而少使用非脚本文件

尽量多使用函数文件而少使用脚本文件，也可以提高执行效率。因为每次调用 MATLAB 的脚本文件都需要将不必要的中间变量加载到内存。函数在调用时被编译成了伪代码，只需要加载到内存一次。因此，当多次调用同一函数时会运行得快一些。

2.8.5 将循环体中的内容转换为 C-MEX

当必须使用 for 循环时，可以考虑将循环体中的语句转换为 C-MEX。C-MEX 是将 M 文件通过 MATLAB 的编译器转换为可执行文件。在 Windows 环境下，它是扩展名为.dll 的动态链接库，可以在 MATLAB 环境下直接执行。这样，循环体中的语句执行时不必每次都解释（interpret）。一般来说，C-MEX 文件的执行速度是相同功能的 M 文件执行速率的 20～40 倍。编写 C-MEX 不同于 M 文件，需要了解 MATLAB C-MEX 规范和更多的耐心。幸运的是，MATLAB 提供了将 M 文件转化为 C-MEX 的工具。怎样将 M 文件通过 MATLAB 编译器转换为 C-MEX 详见 MATLAB 的联机帮助文档或本书的第 4 章。

2.8.6 内存优化

MATLAB 进行复杂的运算需要占用大量的内存。合理地使用内存和提高内存的使用效率，可以提高运行速度，减少对系统资源的占用。

1. 内存管理函数和命令

- clear variablename 从内存中删除名称为 variablename 的变量。
- clear all 从内存中删除所有的变量。
- save 将指令的变量存入磁盘。
- load 将 save 命令存入的变量载入内存。
- quit 退出 MATLAB，并释放所有分配的内存。
- pack 把内存中的变量存入磁盘，再用内存中的连续空间载回这些变量。考虑到执行效率问题，不能在循环中使用。

2. 节约内存的方法

- 避免生成大的中间变量，并删除不再需要的临时变量。
- 当合用大的矩阵变量时，预先指定维数并分配好内存，避免每次临时扩充维数。
- 当程序需要生成大量变量数据时，可以考虑定期将变量写到磁盘，然后清除这些变量。当需要这些变量时，重新从磁盘加载。
- 将矩阵中数据极少时，将全矩阵转换为稀疏矩阵。

2.9 小　　结

本章简要介绍了 MATLAB 的程序设计，侧重于 MATLAB 的编程规范，以提高 M 文件的正确性和通用性。当然，编程规范并不是强制性的。M 文件规范大致 C、C++和 Java 的编程规范相同，有些针对 MATLAB 的特点进行了修改。此外，对 MATLAB 文件评述器和提高 M 文件执行效率的技巧也进行了介绍。在 MATLAB 编程中，只要遵循编程规范并合理利用提高 M 文件执行效率技巧，就可以编写出简捷且高效的代码。

第 3 章　MATLAB 混合编程简介

3.1　进行混合编程的出发点

MATLAB 是一种“草稿纸式”的程序设计语言，它以其强大的计算和绘图功能、大量稳定而可靠的算法库、高效的编程语言和遍布世界各地的庞大用户群，成为数学计算和算法仿真方面事实上的标准。几乎所有的工程计算领域，MATLAB 都有相应的工具箱提供支持，很多第三方软件还在为各种不同的专业开发和完善 MATLAB 的功能。因此，MATLAB 在数值计算领域已处于不可替代的领先地位。

然而，MATLAB 存在的以下缺点限制了它作为通用的软件开发平台：

- MATLAB 作为一种解释性语言，边解释边运行。尽管编程效率极高，但是运行效率却较许多通用的软件开发平台低下，即实现相同功能的代码尽管较 Visual C++、Fortran 等的代码短，但执行时间相对较长。
- MATLAB 语言编写的 M 文件是文本文件，可以用文本编辑器打开，且不能脱离 MATLAB 环境运行。因此，MATLAB 编写的程序不利于程序的保密。
- 尽管 MATLAB 6.0 以后的版本在图形用户接口（GUI）上有很大的改进，但与通用的编程平台相比还是不够灵活。而且，MATLAB 6.5 与硬件接口的能力还较差，不能访问一些底层的硬件设备。

MATLAB 存在的以上缺点限制了它不能作为通用的软件开发平台。而常见的通用编程平台如 Visual Basic, Visual C++和 Fortran 等，尽管功能强大和灵活，但编程效率较低，尤其是当需要快速验证算法时。另外，还存在不同平台之间程序的移植问题，例如 C、Fortran 平台下编写的程序需要在 MATLAB 环境下运行时，如果改写成 M 文件，会加重编程者的负担。

因此，实现 MATLAB 和 Visual C++的混合编程，有助于发挥 MATLAB 和 Visual C++ 各自的优势，降低开发难度，缩短编程时间。幸运的是，MATLAB 的应用程序接口（API）为实现 MATLAB 与 Visual C++等通用编程平台的混合编程提供了可能。

3.2　MATLAB 应用程序接口简介

MATLAB 与外部的数据和程序交互是很有意义的。通过它与其他编程环境的交互，可以扩充 MATLAB 强大的数值计算和图形显示功能，而避开其执行效率较低的缺点。

具体地说，MATLAB 接口包含以下功能。

1. MATLAB Compiler

通过 MATLAB 自带的 C 编译器(Compiler)，可以将其 M 文件转换成为 C/C++代码文件，并生成必要的 DLL，再通过 Visual C++编译器生成可独立执行的应用程序，这种方式可以直接调用其中的库函数，生成并发布不依赖 MATLAB 的可执行文件。当然，并不是所有的 M

文件 MATLAB 编译器都能够将它们编译为可执行程序，在某些情况下，编译器的作用是有限的。为此，MATLAB 提供了运行时服务器（Run time Server），利用它可以很好地弥补编译器的不足。

2. MATLAB 引擎

MATLAB 引擎库是一系列过程，在 C 或 Fortran 等语言中用它们来调用 MATLAB。这样，将 MATLAB 作为一个计算引擎，在后台运行。MATLAB 引擎函数库在用户程序与 MATLAB 进程之间起桥梁作用。MATLAB 提供了一个函数库，用它们启动和终止 MATLAB 进程、传输数据并传递要在 MATLAB 中处理的命令。

3. MAT 数据文件共享数据

MAT 文件是 MATLAB 特定的数据文件格式，MATLAB 提供了一些操作 MAT 数据文件的 API 函数。MATLAB 可以与其他语言平台通过 MAT 文件实现数据共享，从而有助于减轻其他平台的计算负担。

4. 通过 ActiveX

ActiveX 是一类自动化技术的总称。MATLAB 支持组件自动化，对于传统的 DDE 和 ActiveX 自动化，MATLAB 也提供了全面的支持。通过 ActiveX，可以在 MATLAB 和其他软件平台建立客户机/服务器体系结构，方便彼此的交互。

5. Mideva/Matcom

Mideva 是由 MathTools 公司推出的 MATLAB 替代编译环境，提供了将 MATLAB M 文件转换为 C、CPP 源代码、DLL 和 EXE 文件的功能。相对于 MATLAB 自带的编译器，用 Matcom 转化代码要简单和方便得多。但是 Mideva 在很多方面也有限制，比如，对 struct 等类的支持有缺陷，部分绘图语句无法实现或得不到准确图像，尤其是三维图像。

6. MATLAB<LIB>数学库

MATLAB 中提供了大量用 C/C++重新编写的 MATLAB 库函数，包括初等数学函数、线性代数函数、矩阵操作函数、数值计算函数、特殊数学函数、插值函数等，可以直接供 C/C++语言调用。因此，利用 MATLAB 的数学库，可以在 Visual C++中充分发挥 MATLAB 的数值计算功能，并且可以不依赖 MATLAB 软件运行环境。

7. MATLAB Add-in

MATLAB 6.0 以后版本对其 Compiler 进行了较大改进，支持更多的数据类型，更强的优化功能，更重要的是 MATLAB Add-in 提供了一个 MATLAB 和 Visual C++直接集成的途径。

8. MATLAB COM Builder

MATLAB 提供了 COM 生成器。COM（Component Object Model 组件对象模型）是一项比较复杂的技术。COM 生成器提供了实现 MATLAB 独立应用程序的一种新途径。它是把 MATLAB 开发的算法做成组件，这些组件作为独立的 COM 对象，可以直接被 Visual C++、

Visual Basic、Delphi 或其他支持 COM 的语言所引用。MATLAB COM Builder 几乎支持 MATLAB 所有的函数，特别适合在 M 文件较大、使用 Matcom、调用 MATLAB 数字库或使用 MATLAB Compiler 等无法实现时使用。

9. Excel Link

可在 Excel 工作空间和 MATLAB 工作区之间通信，通过链接 Excel 和 MATLAB，可以从 Excel 工作表和宏编程工具中获得 MATLAB 的数值计算和图形绘制功能，能够在两个环境之间交换数据。

10. MATLAB Builder for Excel

MATLAB 还提供了一个称为 Excel 生成器的工具。利用该工具，可以生成 DLL 组件或者 VBA 代码。利用 DLL 组件，可以进行与 COM 生成器组件相似的操作。VBA 代码则可以在 Excel 的 Visual Basic 编辑器中直接使用。

实现 MATLAB 与其他编程语言混合编程的方法很多。通常，根据混合编程时是否需要 MATLAB 运行，可以分为两大类：MATLAB 在后台运行和可以脱离 MATLAB 环境运行。事实上，这些方法各有优缺点，具体使用时需要结合开发者的具体情况。无论使用哪种方法，运行环境的设置以及 MATLAB 的数据结构和语法都是关键内容。

3.3 几种常见的混合编程方法简介

3.3.1 使用 MATLAB 自带的 MATLAB Compiler

自 MATLAB 5.3 以后的版本都自带了 C 语言的编译器（Compiler）。最新的版本是 MATLAB 7.0 版本自带的 Compiler 4.0。MATLAB Compiler 的作用是将 MATLAB M 文件转化成 C/C++代码（也就是通过通常所用的 mcc 命令）。因此，MATLAB Compiler 是 MATLAB 混合编程的基础。生成的 C/C++源代码需要用 C/C++编译器编译链接成独立应用程序。在将 M 文件转成独立应用程序的过程中生成 C/C++文件，原则上它是可以被其他 C/C++代码调用的，编译器可以通过设置 mcc 命令的选项，将 M 文件编译成动态链接库文件、C/C++文件、可执行文件等一系列文件。

图 3-1 是 MathWorks 介绍 MATLAB 混合编程的免费网络课程（Web Seminar）关于 Compiler 的截屏图。MATLAB 有其他很多混合编程方式，实际上都是通过 MATLAB Compiler 实现的，例如后文将介绍到的 MATLAB Add-in 和 MATLAB Builder for COM 等。

最新的 MATLAB Compiler 4.0 版本将 M 程序转换成 C/C++代码功能仍有很多限制，体现在以下几个方面：

- 不能转换脚本 M 文件，只能转换 M 函数；
- 不能使用 MATLAB 对象；
- 不能用 input 或者 eval 操作 MATLAB 空间变量；
- 不能动态地命名变量，然后用 load 或者 save 命令来操作；
- 不能处理具有嵌套调用其他 M 文件的 M 文件；
- 不能使用 MATLAB 内联函数。

MATLAB Compiler

- Automatically converts your MATLAB programs into *stand-alone applications* and *software components*
- Supports full MATLAB language and most toolboxes
- Software components include
  - C and C++ libraries
  - Excel add-ins and COM objects (require MATLAB Builder products)
- Free deployment model

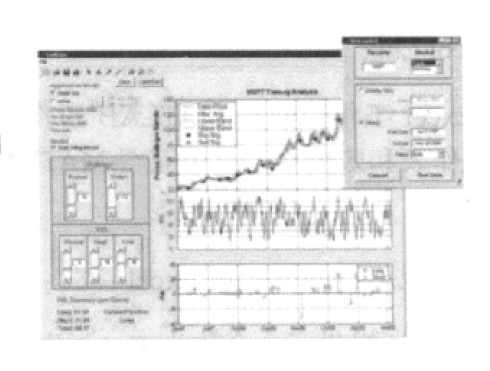


图 3-1　MATLAB 自带的 C 语言编译器

3.3.2　利用 MATLAB 引擎

MATLAB 引擎函数库是 MATLAB 提供的一组接口函数，它允许用户在自己的应用程序中对 MATLAB 函数进行调用。因此，可将 MATLAB 作为一个计算引擎使用，让其在后台运行，这样可以简化前台用户程序的设计任务。

当用户使用 MATLAB 引擎时，采用 C/S（客户机/服务器）模式，相当于在后台启动了一个 MATLAB 进程。MATLAB 引擎函数库在用户程序与 MATLAB 进程之间起桥梁作用，完成两者的数据交换和命令的传送。在 Windows 系统中通过 ActiveX 完成。

MATLAB 引擎（Engine）有以下典型的功能应用：

（1）调用 MATLAB 特有的强大数值计算和分析等函数进行运算。

数值计算如矩阵运算是 MATLAB 的强项，通过 MATLAB 计算引擎可以轻松地实现矩阵运算、图形显示等计算复杂度高的操作。

（2）可以为一个特定的任务构建一个完整的系统。

前台客户机台可以采用诸如 Visual C++之类的通用编程平台，通过 Windows 的动态控件与服务器 MATLAB 通信，向 MATLAB Engine 传递命令和数据信息，从 MATLAB Engine 接收数据信息。此时，MATLAB 完成较复杂的数值计算、分析和可视化任务。因此，MATLAB Engine 可以简化应用程序的开发，取得事半功倍的效果。

MATLAB Engine 几乎可利用 MATLAB 全部功能，但是需要在机器上安装 MATLAB 软件，而且执行效率低，因此这种方式的实用性较低，在商用的应用软件开发中不宜采用。

3.3.3　利用 ActiveX 控件

ActiveX 是一种支持组件集成的 Microsoft Windows 协议，通过 ActiveX 技术可以将不同环境下开发的组件集成到一个应用程序中。ActiveX 同时是面对对象技术的一种，属于基于组件对象模型（COM）的子类。COM 为所有的 ActiveX 对象定义了对象模型，每个 ActiveX 对象支持一定的接口，也就是不同的方法、属性和事件。

MATLAB 支持两种 ActiveX 技术：ActiveX 控制器和 ActiveX Automation。ActiveX 控制

器可以将不同的 ActiveX 控制集成在一个应用中，而 ActiveX Automation 是一种允许一个应用程序（客户端）去控制另一个应用程序（服务器端）的协议。因此，它允许 MATLAB 控制其他 ActiveX 组件或者被其他 ActiveX 所控制。当 MATLAB 控制其他 ActiveX 组件时，MATLAB 作为一个 Automation Client; 当 MATLAB 被其他 ActiveX 控件控制时，MATLAB 就是一个 Automation Server。

3.3.4 利用 MAT 文件

MATLAB 与其他编程环境的数据交互是通过 MAT 数据文件来实现的。MAT 文件是 MATLAB 系统保存文件的默认文件格式，它把数据文件存储为二进制格式。这种格式为不同平台或不同应用程序间共享 MATLAB 数据提供了一种便利的机制。一般情况下，不需要了解 MAT 文件的具体格式，因为 MATLAB 提供了一些 API 函数来简化 MAT 文件的读取与存储，而 MATLAB API 完全屏蔽 MAT 文件格式。因此，在某些情况下，可将计算复杂度高的操作交由 MATLAB 来处理，而在其他应用程序中通过共享 MATLAB 生成的 MAT 数据文件达到减轻编程负担的目的。

3.3.5 C-MEX

所谓 MEX 是 MATLAB Executable 的缩写，即 MATLAB 的可执行程序。在 Windows 环境中，它是扩展名为 DLL 的动态链接库。MEX 文件是 MATLAB 调用其他语言编写的程序或算法的接口。它符合 MATLAB 的调用格式，可以在 M 程序中直接调用。

与 MATLAB 引擎 API 一样，MATLAB 也提供了一组用于 MEX 程序的应用程序接口。在 MATLAB 中，可以调用用户自己开发的 C 或 Fortran 子程序，通过 MATLAB 的 API 函数库将 C 或 Fortran 子程序编译成 MEX 文件，以便在 MATLAB 环境中直接调用或链接这些子程序，达到提高计算效率的目的。

MEX 文件具体有以下几个方面的应用：

- 对于某些已知的 C 或者 Fortran 子程序，可以通过 MEX 方式在 MATLAB 环境中直接调用，而不必重新编写相应的 M 文件。
- 对于影响 MATLAB 执行速度的 for、while 等循环，可以编写相应的 C、Fortran 子程序完成相同的功能，并编译成 MEX 文件，提高运行速度。
- 对于一些需要访问硬件的底层操作，如 A/D、D/A 或中断等，可以通过 MEX 文件直接访问，克服 MATLAB 对硬件访问功能不足的缺点，从而增强 MATLAB 应用程序的功能。

当然，并不是所有的程序都适合用 MEX 实现。MATLAB 编程效率极高，因此一般在 MATLAB 环境编程时尽量以 MATLAB 编程为主，只是对那些耗时或者 MATLAB 功能受限的部分采用 MEX 编程。

3.3.6 利用 Mideva/Matcom

Mideva 是 MathTools 公司推出的一种 MATLAB 编译开发软件平台，提供对 MATLAB 程序文件（M 文件）的解释执行和开发环境支持。该软件有为 Borland C++、Visual C++和 Visual Basic 等编程语言开发的不同版本，目前其版本已经到了 4.5 版。软件仅为 6.5MB，可以通过访问其站点 www.mathtools.com 免费下载试用一个月。

Mideva 是一个集成的调试编辑环境，提供了比 MATLAB 更强大的编辑、调试功能，如语法突出显示、批量注释等。而 Matcom 是 Mideva 的内核，它是一个基于 C++矩阵函数库 Matrix 的一个 MATLAB M 文件与 CPP 文件的转换程序。

Mideva 提供的功能相当强大，因为它包含了近千个 MATLAB 的基本功能函数，通过必要的设置就可以直接实现与 C++的混合编程，而不必再依赖 MATLAB。同时，Mideva 还提供编译转换功能，能够将 MATLAB 函数或编写的 MATLAB 程序转换为 C++形式的 DLL，从而实现脱离 MATLAB 环境对 MATLAB 函数和过程的有效调用，这样就有可能实现对 MATLAB 强大工具箱函数的利用。Mideva 不仅可以转换独立的脚本文件，也可以转换嵌套脚本文件。

Matcom 的不足之处在于对 struct 等类的支持有缺陷，而且部分绘图语句无法实现或得不到准确图像，尤其是三维图像。因此，在不涉及到三维绘图以及 M 文件不太大的情况下适合使用。

3.3.7 利用 Matrix<LIB>实现混合编程

Mideva/Matcom 提供一种实现混合编程的方法，它自动将 M 文件转换为 C、CPP 文件，然后将生成的 C、CPP 代码拷贝到 Visual C++工程中，从而可降低开发难度。实际上，Matcom 的内核是一组称为 Matrix<LIB>的 C++库。它是一个矩阵数学库，提供了一个双精度 Matrix 类型——M<double>，它可以是复数矩阵、实数矩阵、稀疏矩阵甚至 N 维矩阵。这个库共有 500 多个函数，涉及线性代数、多项式数学、信号处理、文件输入/输出、图像处理、绘图等方面。

其实我们可以在 Visual C++工程中直接按照 Matrix<LIB>的语法编程，即可实现高效的编程。这是因为 Matrix<LIB>的函数是基于矩阵类型的，大多数函数原型类似于 MATLAB 函数。实际上，Matrix<LIB>是将一些很有用的 MATLAB 函数封装成 C++的库文件。它要求读者对 C/C++语言比较熟悉，适合对 C/C++语言比较熟悉的用户使用。

3.3.8 利用 MATLAB Add-in

MathTools 公司被 MathWorks 公司收购后，将 Visual Matcom 集成到了 MATLAB 6.0 版本中。因此，MATLAB 6.0 以后版本对其 Compiler 进行了较大的改进，支持更多的数据类型、更强的优化功能，更重要的是其 MATLAB Add-in 提供了一个 MATLAB 和 Visual C++直接集成的途径。

MATLAB Add-in 具有以下一些新的特征：

- 快速地集成 M 文件到 C++的工程中，可创建独立的 C/C++应用程序或 C MEX DLL；
- 通过 M 文件创建共享库或 MEX 文件；
- 内含 Visual Matrix Viewer，调试过程中可以查看矩阵变量的值；
- 直接修改 M 源文件而不是修改生成的 C/C++文件，方便快捷地打包应用程序等。

MATLAB 7.0 版本自带的编译器 Compiler 4.0 不再支持这种方式，主要原因是其生成的 C/C++代码不够直观，用户通常难以修改。

3.3.9 MATLAB COM Builder

COM 是以组件为发布单元的对象模型。由于 COM 是建立在二进制级别上的规范，所以

组件对象之间的交互规范不依赖于任何特定的开发语言。COM 适合于不同语言之间的协作开发。COM 开发架构是以组件为基础的，因而可以把组件看做用于“搭建”软件的积木。

MATLAB 提供了 COM 生成器。COM 生成器提供了实现 MATLAB 独立应用程序的一种新途径。它是把 MATLAB 开发的算法做成组件，这些组件作为独立的 COM 对象，可以直接被 Visual C++、Visual Basic、Delphi 或其他支持 COM 的语言所引用，可以生成不依赖 MATLAB 环境的独立程序，因此可获得最快的运行速度，不需进行代码转换，使得编程风格一致，可读性好。图 3-2 是 MathWorks 介绍 MATLAB 混合编程的免费网络课程关于 MATLAB Builder for COM 的截屏图。

实际上，COM 生成过程实际是上通过 MATLAB 的 C 语言编译器 mcc 生成的。其实，采用 mcc 和 mbuild 命令完全可以实现 MATLAB COM Builder 的全部功能，只不过 MATLAB COM Builder 为用户提供了一组更加友好的图形用户界面。

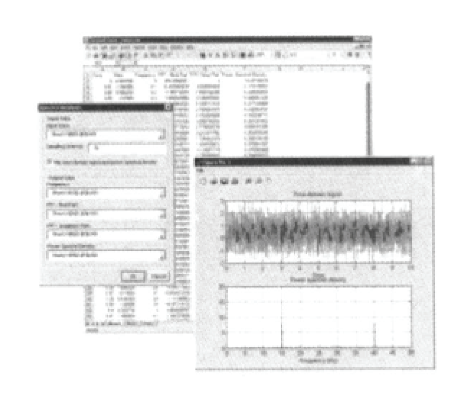


图 3-2 MATLAB Builder for COM 功能

3.3.10 MATLAB 和 Excel 混合编程

Excel 和 MATLAB 在图形显示和数值计算上各有优势。因此，MATLAB 提供了 Excel Link 和 MATLAB Builder for Excel 两个工具实现两者的混合编程。

Excel Link 是一个在 Windows 环境下实现 MATLAB 和 Excel 相互链接的插件。通过对 Excel 和 MATLAB 的链接，用户可以在 Excel 工作区间里，利用 Excel 的宏编程工具，使用 MATLAB 的数据处理和图形处理功能进行相关操作，同时由 Excel Link 来保证两个工作环境数据的交换和同步更新。使用 Excel Link 时，不必脱离 Excel 环境，而是直接在 Excel 工作区或者宏操作中调用 MATLAB 函数。

MATLAB Builder for Excel 是 MATLAB 专门为 Excel 提供的生成器工具，能够轻易地将复杂的 MATLAB 算法转换为独立的 Excel 加载宏（插件）。使用者可以像使用其他 Excel 加载宏功能一样地参照 MATLAB 算法。MATLAB 所建立的 Excel 加载宏函数依赖计算功能强大的 MATLAB 函数库，其运算速度可以比利用 VBA 所建立的 Excel 加载宏快许多。从 MATLAB 所建立的 Excel 加载宏函数也可以免费分享至任何 Excel 应用程序。利用该工具，可以生成 DLL 组件或者 VBA 代码。利用 DLL 组件，可以进行与 COM 生成器组件相似的操

作。VBA 代码则可以在 Excel 的 Visual Basic 编辑器中直接使用。

3.4 小　　结

本章在分析 MATLAB 与其他高级语言混合编程出发点的基础上，对 MATLAB 应用程序接口进行了简要介绍，重点说明了 MATLAB 的几种常见混合编程实现方法。希望通过本章的学习，对 MATLAB 的混合编程有一个总体上的认识。

第 4 章　C-MEX 编程

4.1　C-MEX 简介

MEX 文件是按一定格式、使用 C 语言或 Fortran 语言编写、由 MATLAB 解释器自动调用并执行的动态链接库（Dynamic Link Library, DLL）函数。在 Windows 系统中，这种文件类型的后缀名为.DLL。

MEX 文件的使用极为方便，只需在 MATLAB 命令提示符下键入 MEX 文件名即可，这与 MATLAB 内在函数的调用方式完全相同。当用户执行一个 MEX 文件时，MATLAB 系统将首先搜索 MATLAB 系统的所有可搜索路径，然后载入并执行第一个与用户键入的文件名相匹配的可执行文件。因此，在 MATLAB 中存在两种类型的可执行文件：MEX 文件和 M 文件。当系统中存在名称相同的两种类型的可执行文件时，即 MEX 文件和 M 文件都存在时，MATLAB 系统规定，MEX 文件的执行优先级高于 M 文件，这样 MEX 文件将优先执行。

此外，由于 MEX 文件没有提供应用的帮助信息。一般情况下，每当构造一个 MEX 文件，就应当编写一个 MATLAB 的 M 文件，作为相应的帮助文件，并且存放在同一目录下。在该 M 文件中，不包含任何可执行语句，只是包含一些帮助信息，用来对同名的 MEX 文件的功能及输入/输出参数进行说明。因此，用户就可以在 MATLAB 提示符下，通过使用 help 命令获取帮助。在 MATLAB 的解释器中，help 命令仅仅对与命令同名的 M 文件进行查找，而忽略对 MEX 文件的查找。

MATLAB MEX 文件是 MATLAB 系统的外部程序调用接口。通过它，用户可以完成以下功能：

可以在 MATLAB 系统中像调用 MATLAB 的内在函数一样调用已经存在的用 C 语言和 Fortran 语言编写完成的算法，通过添加入口程序 mexFunction，而无须将这些函数重新编写为 MATLAB 的 M 文件，从而使资源得到充分利用。

当使用 MATLAB 进行大规模的数据处理时，MATLAB 往往由于执行效率的问题而显得力不从心，这时可以使用其他高级编程语言进行算法的设计，然后在 MATLAB 环境中调用，从而大大提高数据处理的效率。在 MATLAB 中，M 文件的计算速度特别是循环迭代的速度远比 C 语言慢，因此可以把要求大量循环迭代的部分用 C 语言编写为 MEX 文件，提高计算速度。

通过 MEX 文件，用户可以直接对硬件进行编程，如 A/D 采集卡，D/A 输出卡等，以用于数据采集或控制，进一步拓展 MATLAB 的应用领域。

4.2　MEX 文件系统的配置

MEX 文件的编写与编译需要同时具备两个条件：一是要求已经安装 MATLAB 应用程序接口组件及其相应的工具，另一个是要求有合适的 C 或 Fortran 语言编译器。另外，如果是

在 Windows 平台上，那么用户的编译器需要能够支持 32 位的 Windows 动态链接库。本书主要以 Visual C++ 6.0 为外部的 C 语言编译器，这些要求是满足的。因此，接下来的工作是对 MATLAB 系统进行配置，以便让 MATLAB 系统知道使用哪一个编译器以及编译器的参数情况。

编译器的配置文件是系统配置过程中产生记录配置信息的文件，后缀为.bat，是一个批处理文件。编译器的配置文件包括一些关于编译、预编译和链接阶段的特殊标志和变量，这些标志和变量标示编译过程的每一个逻辑步骤。

在 Windows 系统中编译 MEX 文件，要先对 MEX 文件编译器的默认配置文件 mexopts.bat 进行配置编译。编译器的配置命令为：

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:
```

Would you like mex to locate installed compilers [y]/n? 提示用户是否定位编译器，选择“y”，则 MATLAB 继续输出以下信息：

```
Select a compiler:
[1] Digital Visual Fortran version 6.0 in C:\Program Files\Microsoft Visual Studio
[2] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual Studio
[0] None
```

Compiler: 提示用户选择编译器，通常输入“3”，选择 Visual C++。

```
Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0
Location: C:\Program Files\Microsoft Visual Studio
```

Are these correct? ([y]/n): 提示系统找到的 Visual C++安装路径是否正确，通常系统可以找到正确的路径，因此输入“y”。

```
Try to update options file: C:\Documents and Settings\Administrator\Application
Data\MathWorks\MATLAB\R14\mexopts.bat
From template: C:\MATLAB7\BIN\WIN32\mexopts\msvc60opts.bat

Done . . .
```

至此，整个配置过程完成。

4.3 MEX 文件的结构和运行

4.3.1 MEX 文件结构

下面以 MATLAB 软件自带一个 C-MEX 文件为例，说明 MEX 文件的结构。位于“<MATLABroot>\extern\example”目录下的 timestwo.c 结构清晰，体现了 C-MEX 文件的基本框架。现将 timestwo.c 列举如下（略有删减，中文注释为编者添加）：

```
#include "mex.h"
/*
 * timestwo.c - example found in API guide
 *
 * Computational function that takes a scalar and double it.
 *
 * This is a MEX-file for MATLAB.
 * Copyright 1984-2000 The MathWorks, Inc.
 */
 /* $Revision: 1.8 $ */

//本 MEX 文件的目的是实现 timestwo 的功能
void timestwo(double y[], double x[])
{
   y[0] = 2.0*x[0];
}

//下面这个 MexFunction 的目的是使 MATLAB 知道如何调用这个 timestwo 函数
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
//nlhs: MATLAB 命令行方式下输出参数个数
//plhs[]: MATLAB 命令行方式下输出参数
//nrhs: MATLAB 命令行方式下输入参数个数
//prhs[]: MATLAB 命令行方式下输入参数
double *x,   *y;
   int mrows, ncols;

   //检查输入、输出变量个数
   if(nrhs!=1) {
      mexErrMsgTxt("One input required.");
   } else if(nlhs>1) {
      mexErrMsgTxt("Too many output arguments");
   }

   //输入必须是一个非复数浮点类型矩阵

   //获得输入矩阵的行、列数
   mrows = mxGetM(prhs[0]);
   ncols = mxGetN(prhs[0]);

   //判断输入矩阵是否是 double 类，以及它是否包括单个元素
```

```
    if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
        !(mrows==1 && ncols==1) ) {
      mexErrMsgTxt("Input must be a noncomplex scalar double.");
    }

    //为输出创建一个矩阵，显然这个矩阵也应该是1×1的
    plhs[0] = mxCreateDoubleMatrix(mrows,ncols, mxREAL);

    //获得指向输入、输出矩阵数据的指针
    x = mxGetPr(prhs[0]);
    y = mxGetPr(plhs[0]);

    //调用C++函数 timestwo(y,x)
    timestwo(y,x);
}
```

可见，C 语言 MEX 文件的源程序主要由两个截然不同的部分组成，分别用于完成不同的任务：第一部分称为计算子程序，它包含了所有实际需要完成计算功能的源代码，用来完成实际的计算工作，即为用户以前所编写的算法和程序；由于它是以函数形式存在的，所以如果用户想要将一些已经编写的程序和算法移植到 MATLAB 环境中使用，就需要将算法和程序整理成函数形式封装。第二部分称为入口子程序，它是计算子程序同 MATLAB 环境之间的接口，用来完成两者之间的通信任务。它定义被 MATLAB 调用的外部子程序的入口地址、MATLAB 系统向子程序传递的子程序参数、子程序向 MATLAB 系统返回的结果参数，以及调用计算功能子程序等。

入口子程序的名字为 mexFunction，它的原型在 mex.h 中定义为：

```
void mexFunction(
    int             nlhs,           /* number of expected outputs */
    mxArray         *plhs[],        /* array of pointers to output arguments */
    int             nrhs,           /* number of inputs */
    const mxArray *prhs[]           /* array of pointers to input arguments */
);
```

函数 mexFunction 共有 4 个参数，分别为 prhs,nrhs, plhs 和 nlhs，其中 prhs 为一个 mxArray 结构体类型的指针数组，该数组的数组元素按顺序指向所有的输入参数；nrhs 为整数类型，它标明了输入参数的个数；plhs 同样为一个 mxArray 结构体类型的指针数组，该数组的数组元素按顺序指向所有的输出参数；nlhs 则标明了输出参数的个数，为整数类型。这些参数是用来传递 MATLAB 启动 MEX 文件的参数。

在入口子程序中，用户主要可以完成两个方面的任务：一方面，从输入的 mxArray 结构体中获得计算所需的数据，然后在用户的计算子程序中加以使用。另一方面，用户同样可以将计算完毕的结果返回给一个用于输出的 mxArray 结构体，这样 MATLAB 系统就能够识别从用户计算子程序返回的结果。

MEX 源文件的两个组成部分既可以存放在一个文件中，也可以分为两个文件来存放，这无关紧要，重要的是文件必须对头文件 mex.h 进行包含，因为该头文件中不但包含了最基本

的头文件 matrix.h（定义了矩阵），而且包含了所有以 mex 为前缀的库函数的声明。不管怎样，入口子程序必须是 mexFunction，其构成形式为：

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
//一些必要的C语言代码，用来完成MATLAB与计算子程序的通信任务
}
```

4.3.2 MEX 函数的执行流程

当对 C 语言 MEX 文件进行编译时，首先 MATLAB 从命令行获取输入矩阵，并赋给入口子程序的输入参数指针组，入口子程序对输入参数进行简单的检查后，就创建输出矩阵，并将输出矩阵赋给入口子程序的输出参数指针组，然后使用入口子程序的输入参数和输出参数作为计算子程序的参数调用计算子程序。在入口子程序中完成以下任务：

使用 mxGet 函数从 prhs[0], prhs[1], …中提取数据，使用 mxCreat( )函数为输出参数创建矩阵，并将指针 plhs[0], plhs[1], …指向新创建的矩阵，以输入/输出数据的指针作为参数调用计算子程序。图 4-1 显示了如何向 MEX 文件输入数据，通过接口函数调用计算程序完成运算过程，最后向 MATLAB 返回计算结果。

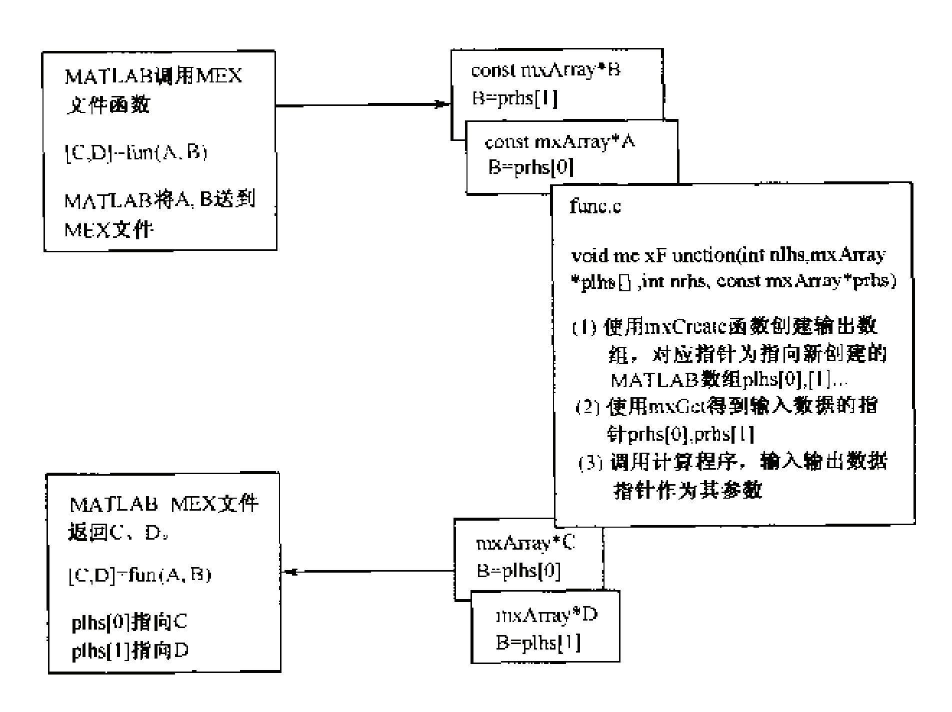


图 4-1 MEX 文件的编译流程图

按 MATLAB 的语法规则，函数（包括 MEX 文件）调用的一般形式是：

```
[a, b, c, …]=fun(d, e, f, …)
```

这里，fun 是 MEX 文件名或 M 文件名。"…" 表示参数的个数可以是任意的，变量 a, b, c 为输出变量，而 d, e, f 为输入变量。

例如，如果要调用一个名为 funmex 的 MEX 文件，则在 MATLAB 环境中其命令形式为：

```
[u, v]=funmex(x, y, z)
```

随后，MATLAB 从 funmex 的 mexFunction 处开始执行，且参数传递如下：

```
nlhs=2;
nrhs=3;
plhs=(指针→'NULL')
(指针→'NULL')
prhs=(指针→x)
     (指针→y)
     (指针→z)
```

在执行子程序调用之前，由于输出变量 u 和 v 对象还没有创建，所以参数 plhs 的各指针指向的地址暂时未定义。在入口程序中，先根据需要创建 mxArray（或 Matrix）类型的输出变量，其地址分别保留在 plhs[0]和 plhs[1]中，再通过两个指针，就可以在计算子程序中使用这些变量进行计算。

4.3.3 MEX 文件的结构和使用

在 C 语言的 MEX 文件中，参数 nlhs 和 nrhs 包含输出和输入变量的数目，借助这两个参数，MEX 文件被调用。参数 plhs 和 prhs 是包含指向 MEX 文件输出和输入变量的指针的向量，prhs 是长度为 nrhs 的输入变量的指针数组，plhs 是长度为 nlhs 的输出变量的指针数组。例如，从 MATLAB 命令窗口来调用一个 MEX 文件：

```
x=fun(y, z);
```

则 MATLAB 编译器使用下面的变量来调用 mexFunction：

```
nlhs=1
nrhs=2
plhs=(pointer)→/*unassigned*/
prhs=(pointer)→y
(pointer)→z
```

plhs 指向只有一个元素的 C 语言数组，并且这个元素为空指针。prhs 指向含有两个元素的 C 语言数组，其中第一个元素指向 mxArray 型变量 y，第二个元素指向 mxArray 型变量 z。

这里，plhs 指向空的数组是由于输出 x 在子程序执行前尚未产生，入口程序的作用就是创建输出数组并分配指针 plhs[0]指向该数组。如果 plhs[0]没有赋值，MATLAB 将给出输出变量没有赋值的警告信息。

4.3.4 MEX 文件与独立应用程序的区别

MEX 文件和独立应用程序在以下方面存在差异：

MEX 文件与 MATLAB 解释器工作于同样的过程空间。当启动一个 MEX 文件时，MATLAB 解释器动态链接 MEX 文件。也就是说，MEX 文件不能脱离 MATLAB 环境运行。

独立应用程序可以脱离 MATLAB 环境运行。可以在独立应用程序中调用 C-MEX 文件。

4.4 C 语言 MEX 函数

C 语言 MEX 函数是 MATLAB 应用程序接口函数库提供的一种库函数，均以 mex 为前缀，主要功能是与 MATLAB 环境进行交互，例如从 MATLAB 环境获取必要的阵列数据或者返回一定的信息。这种类型的库函数只能用于 MEX 文件中，它们的原型都定义在 <MATLABroot>\extern\include 目录下的 mex.h 中。下面对 mex.h 中定义的 C-MEX 函数的原型和功能进行详细说明，对每个函数的参数说明则从略。

1. mexErrMsgTxt( )

原型：void mexErrMsgTxt(const char *error_msg);
功能描述：发出报错信息，然后返回 MATLAB 提示符。

2. mexErrMsgIdAndTxt( )

原型：void mexErrMsgIdAndTxt(const char * identifier, const char * err_msg,...);
功能描述：按指定的格式发出报错信息，并返回 MATLAB 提示符。

3. mexWarnMsgTxt( )

原型：void mexWarnMsgTxt(const char *warn_msg);
功能描述：触发一个没有定义标识的告警信息。

4. mexWarnMsgIdAndTxt( )

原型：void mexWarnMsgIdAndTxt (const char * identifier, const char * warn_msg,...);
功能描述：按指定的格式发出告警信息，并返回 MATLAB 提示符。

5. mexPrintf( )

原型：int mexPrintf(const char *fmt,...);
功能描述：相当于 MATLAB 中的 disp 命令。在 mex.h 中有以下语句，将 printf 语句转为按 mexPrintf 执行。

```
#define printf mexPrintf
```

6. mexMakeArrayPersistent( )

原型：void mexMakeArrayPersistent(mxArray *pa);
功能描述：从 MATLAB 的内存分配表中删除矩阵的所有组件，包含矩阵头本身。当删除一个矩阵时必须调用 mxDestroyArray( )函数。

7. mexMakeMemoryPersistent( )

原型：void mexMakeMemoryPersistent(void *ptr);
功能描述：从 MATLAB 的内存分配表中去除前面通过 mxAlloc( )分配的内存。为了释放内存，需要调用 mxFree( )函数。

8. mexGetFunctionHandle( )

原型：void mexGetFunctionHandle(void);
功能描述：查找一个函数，并返回可与 mexCallMATLABFunction 一起使用的函数句柄。

9. mexCallMATLABFunction( )

原型：void mexCallMATLABFunction(void);
功能描述：执行一个函数，函数句柄是通过 mexGetFunctionhandle 得到的。

10. mexRegisterFunction( )

原型：void mexRegisterFunction(void);
功能描述：记录一个函数指针，作为 MATLAB 可调用函数。

11. mexSet( )

原型：int mexSet(double handle, const char *property, mxArray *value);
功能描述：相当于 MATLAB 的 set 函数，用于设置句柄的属性值。

12. mexGet( )

原型：const mxArray *mexGet(double handle, const char *property);
功能描述：相当于 MATLAB 的 get 函数，用于返回句柄的属性值。

13. mexCallMATLAB( )

原型：int mexCallMATLAB(int nlhs,mxArray *plhs[],int nrhs,mxArray *prhs[],const char *fcn_name);
功能描述：调用 MATLAB 函数，参数的输入/输出参数在其中。

14. mexSetTrapFlag( )

原型：void mexSetTrapFlag(int flag);
功能描述：设置或清除 MATLAB 的陷阱标志。

15. mexSubsAssign( )

原型：void mexSubsAssign(mxArray *plhs,const mxArray *prhs,const mxArray *subs[],int nsubs);
功能描述：执行数组的下标操作，即对数组中某些元素进行操作。其中 subs 为下标数组，nsubs 为下标的个数。

16. mexSubsReference( )

原型：mxArray *mexSubsReference(const mxArray *prhs,const mxArray *subs[], int nsubs);
功能描述：检索一个矩阵的子集，即子矩阵。

17. mexPrintAssertion( )

原型：void mexPrintAssertion(const char *test, const char *fname, int linenum, const charint);
功能描述：输出执行 Assert 时的出错信息，返回控制至 MATLAB 命令行方式。

18. mexGetVariablePtr( )

原型：const mxArray *mexGetVariablePtr(const char *workspace,const char *name);
功能描述：返回指向指定工作区间的指定变量的矩阵的指针。

19. mexPutVariable( )

原型：mexPutVariable(const char *workspace,const char *name,const mxArray *parray);
功能描述：在指定的工作区创建变量，并将矩阵的值赋给它。

20. mexLock( )

原型：void mexLock(void);
功能描述：锁定一个 MEX 函数，以便它不能从内存中清除。

21. mexUnlock( )

原型：void mexUnlock(void);
功能描述：解除对一个 MEX 函数的锁定，以便它可以从内存中清除。

22. mexIsLocked( )

原型：bool mexIsLocked(void);
功能描述：返回 MEX 函数的锁定状态，即是否锁定。

23. mexFunctionName( )

原型：const char *mexFunctionName(void);
功能描述：返回当前正在执行的 MEX 函数的名称。

24. mexEvalString( )

原型：int mexEvalString(const char *str);
功能描述：分析字符串中指定的 MATLAB 命令的语法并执行。如果执行成功则返回零，如果失败则返回非零值。

25. mexAtExit( )

原型：int mexAtExit(void(*exit_fcn)(void));
功能描述：记录 MEX 文件的出口函数。

另外，还有一些函数如 mexGetGlobal( )、mxSetString( )、mxSetDispMode( )、mexGetMatrixPtr( )、mexGetMatrix( )、mexPutMatrix( )、mexPutFull( )、mexGetFull( )、mexGetEps( )、mexGetInf( )、mexGetNaN( )、mexIsFinite( )、mexIsInf( )、mexIsNaN( )、

mexPutArray( )、mexGetArray( )、mexGetArrayPtr( )、mexAddFlops( )等已经过时，一般不再使用。它们被一些新的函数所取代，例如函数 mexPutArray( )被函数 mexPutVariable( )所取代，函数 mexGetArray( )被函数 mexGetVariable( )所取代，而函数 mexGetArrayPtr( )被函数 mexGetVariablePtr( )所取代。

4.5 C-MEX 混合编程

MATLAB 5 API 提供了一系列程序来处理 MATLAB 所支持的各种数据类型，每一种数据类型都有对应函数来处理对应的数据。下面给出一个简单的 C 语言程序和与之对应的 MEX 文件代码，该程序的功能是将标量 x 加倍。

C 语言程序如下：

```
#include "math.h"
void timestwo(double y[], double x[])
{
  y[0]=2.0*x[0];
  return;
}
```

下面是与 C 语言程序功能相同的 MEX 文件：

```
#include "mex.h"
void timestwo(double y[], double x[])
{
  y[0]=2.0*x[0];
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  double *x, *y;
  int mrows, ncols;

/*检查正确的参数数目*/
  if(nrhs!=1)
  {
        mexErrMsgTxt("需要一个输入参数.");
  }
  else
        if(nlhs>1)
        {
              mexErrMsgTxt("输出参数太多.");
        }
```

```
/*输入变量必须是非复数类型的标量*/
mrows=mxGetM(prhs[0]);
ncols=mxGetN(prhs[0]);

if(!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) || !(mrows==1 && ncols==1))
{
      mexErrMsgTxt("输入变量必须是非复数类型的标量.");
}

/*为返回参数创建矩阵*/
plhs[0]=mxCreateDoubleMatrix(mrows, ncols, mxREAL);

/*分配输入/输出参数的指针*/
x=mxGetPr(prhs[0]);
y=mxGetPr(plhs[0]);

/*调用 timestwo 子函数*/
timestwo(y, x);
}
```

C 语言程序是在编译时检查函数参数。MATLAB 可以在 M 函数中传递任意数量和类型的参数，MEX 文件也是如此，不过在程序中必须可靠地处理输入/输出参数的数目。如果将上面的 MEX 文件命名为 timestwo.c，就可以对它进行编译和链接，在 MATLAB 命令窗口中输入：

```
mex timestwo.c
```

这是产生 MEX 文件 timestwo.***所必需的步骤，该文件的扩展名由所运行的系统平台的类型确定。在 Windows 系统下，扩展名为 dll。此时就可以像调用 M 函数一样调用 timestwo 了。

在 MATLAB 命令窗口中输入：

```
>> x=2;
>> y=timestwo(x);
```

就可以得到：

```
y=4;
```

4.6 Visual C++中 MEX 文件的建立和调试

MATLAB 6.5 提供了通过 Add-in 生成和调试 C-MEX 程序的能力。方法是先通过 MATLAB Project Wizard 建立一个新工程，如图 4-2 所示。然后，在 step 1 of 1 的 Visual MATLAB Application Type 选项中选择 C-MEX DLL，如图 4-3 所示。通过 MATLAB Add-in，可以将 MATLAB 程序编译为 mex 程序，但是这一工具还不完善。一个最明显的问题是，不能以 CPP 文件的形式转换 M 程序并进行编译，而生成 C 程序就可以正常转换和编译。这个

问题也是 MathWorks 公司官方承认的。实际上，MATLAB Project Wizard 所能完成的工作，都可以在 MATLAB 命令行方便地实现。因此，通常情况下不需要 Visual C++的集成环境。

但是在有的场合，如果 MEX 程序比较复杂，就可能会考虑利用 Visual C++集成环境的特性对代码进行管理。例如，程序中有些与操作系统密切联系的代码，包括通过 Windows API 显示一个标准的 Windows 对话框，或者对系统硬件进行访问和控制等，而最多的情况就是工程的方式对代码进行多文件的管理。

在编制这些复杂的 MEX 程序时，利用 Visual C++的 Project Wizard 和 Class Wizard 生成 MEX 程序的框架，并用 MFC 类库进行界面设计编程，实现和管理起来就比较方便，不过还要进行一些简单的设置。

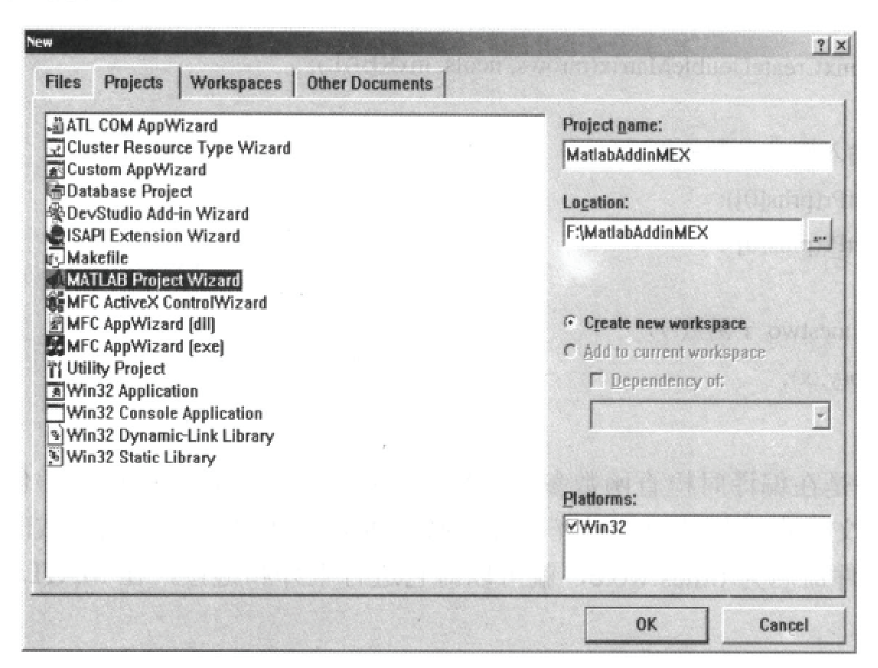


图 4-2　通过 MATLAB Add-in 集成到 Visual C++工程中的 MATLAB Project Wizard

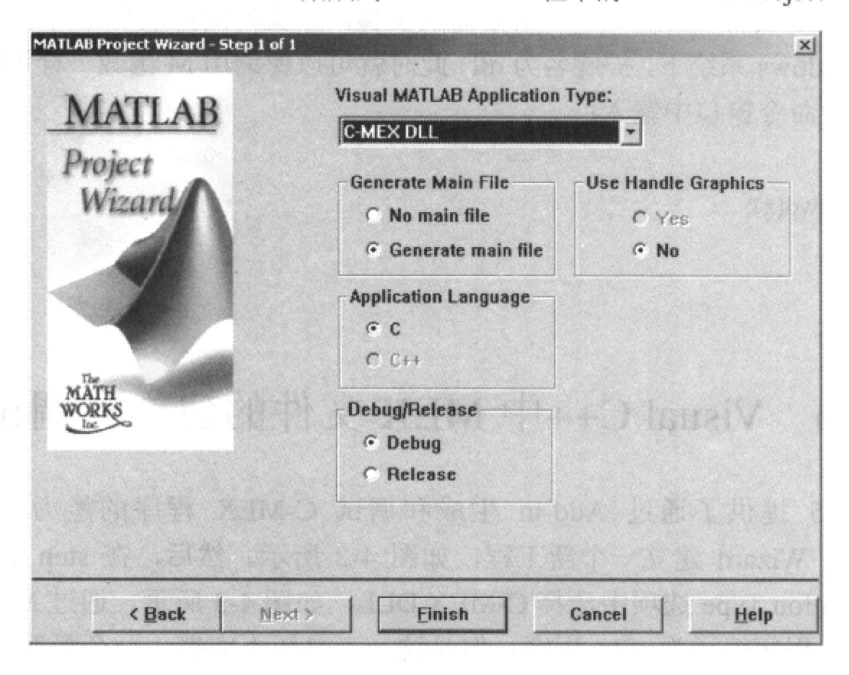


图 4-3　通过 MATLAB Project Wizard 生成 C-MEX DLL 文件窗口

4.6.1 Visual C++中 MEX 程序的建立和环境设置

一个 MEX 程序实际上就是一个特别的动态链接库（DLL），它的输出函数是 mexFunction( )。在 Visual C++工程中可以通过 Project Wizard 建立一个 DLL 工程，并为其指定输出函数为 mexFunction( )。

4.6.1.1 建立一个新的 DLL 工程

启动 Visual C++，选择“File|New”选项，在 Project 页面中选择 MFC AppWizard(dll)，输入路径名和工程名，并单击“OK”按钮。

4.6.1.2 选择静态链接的 MFC 类库

在 MFC AppWizard 的 Step 1 of 1 对话框中，选择使用静态链接 MFC 的 DLL，如图 4-4 所示。注意不要选择动态链接的 MFC 库。因为 MATLAB 使用了特殊版本的 DLL 文件 mfc42.dll，并且放置在其程序目录中，MATLAB 必须使用自带的 mfc42.dll，而这个文件与 Windows 系统目录中的相应文件是不兼容的，若替换则将会导致 Windows 系统或 MATLAB 工作不正常，严重时甚至造成 MATLAB 被非法操作而异常退出。

当 MATLAB 启动需要 mfc42.dll 时，会首先在 MATLAB 所在的程序目录中搜索，并找到这个特殊版本的 DLL 文件，而不会用 Windows 系统目录中的。其他应用程序则根据系统路径中的设置，找到在 Windows 系统目录中的 mfc42.dll，因此不会互相干扰。如果在建立 MEX 工程时选择了动态链接 MFC 库，它将调用位于 MATLAB 程序目录中的 mfc42.dll，由于版本不兼容的问题，在某些情况下会出错，而选择静态链接库就可以避免出现这个问题。

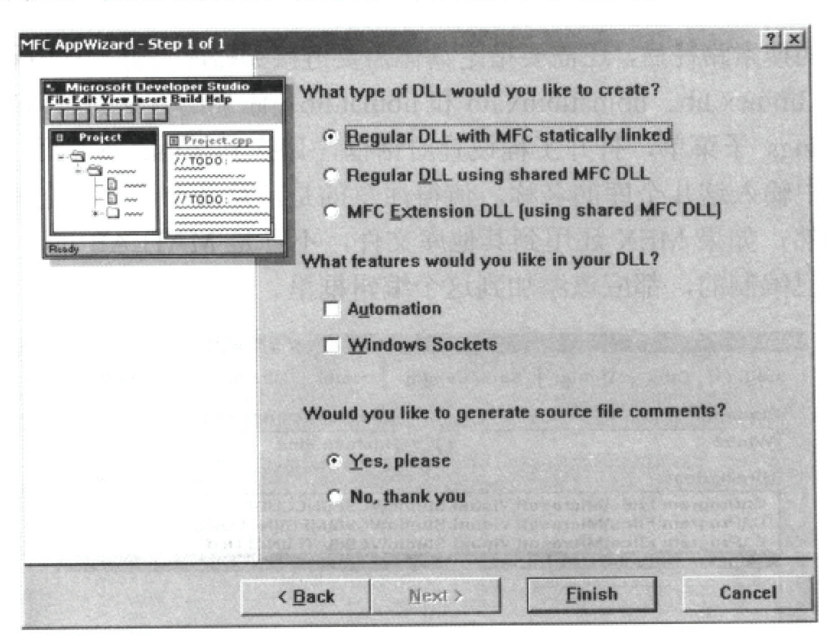


图 4-4 选用静态链接的 MFC 类库

4.6.1.3 设定输出函数

通过以上步骤，将会在工程中看到一个 DEF 文件。需要在其中指定 DLL 文件的输出函数，也就是 mexFunction( )。为此，在 Visual C++的 Workspace 栏中，单击“FileView”属性

页，展开“Source Files”栏，打开该 DEF 文件，并对其内容进行编辑，即在“Exports”后面加入 mexFunction( )一行。

4.6.1.4 Visual C++中必要的环境设置

由于在编译和链接 MEX 程序的时候，需要用到 MATLAB 提供的若干个头文件和库文件，因此应该对这些文件的搜索路径进行指定。在 MATLAB 6.0 以前的版本中，没有提供必要的库文件（LIB），只提供了相应的 DEF 文件，必须由用户用“lib”命令手工将 DEF 文件转换为 LIB 文件。在 MATLAB 6.0 以后版本中，则直接提供 LIB 文件，不需要用户生成了。

1. 添加库函数和头文件必要的路径

MEX 文件所需用到的头文件（如 mex.h 和 matrix.h 等）位于 MATLAB 的<MATLABroot>\extern\include 目录下。而 MATLAB 对不同版本的编译器提供了不同的库文件，分别存放在不同的目录下。对于 Visual C++ 6.0 而言，库文件在 MATLAB 的<MATLAB>\extern\lib\win32\Microsoft\msvc60 目录下。

因此，需要将这两个目录添加到 Visual C++的搜索路径中。方法是通过选择菜单“Tools|Options”，打开选项对话框，在 Directories 属性页中，单击 Show directories for 选项，在下拉列表中选择 Include files，在列表中添加 MATLAB 头文件的目录（如图 4-5 所示，蓝色部分为新添加，既可以直接输入也可通过 browse 添加），然后在下拉列表中选择 Library files，在列表中添加 MATLAB 库文件的目录（如图 4-6 所示）。

2. 添加必要的库文件

指定库文件的搜索路径后，还需要指定具体需要链接哪些库文件。常用到的库文件有以下几个：libmx.lib、libmex.lib、libmatlbmx.lib 和 libmat.lib。添加的方法如图 4-7 所示，通过选择 Project 下的 Settings 子菜单，打开工程设置对话框，切换到 Link 属性页，在 Object/library modules 编辑框中输入这几个库的名称。值得注意的是，这几个库文件之间通过空格隔开，否则会报错。当然，如果 MEX 还用到其他库文件，不管是 MATLAB 的，还是 Windows 系统的，或者是自己编制的，都应该添加到这个编辑框里。

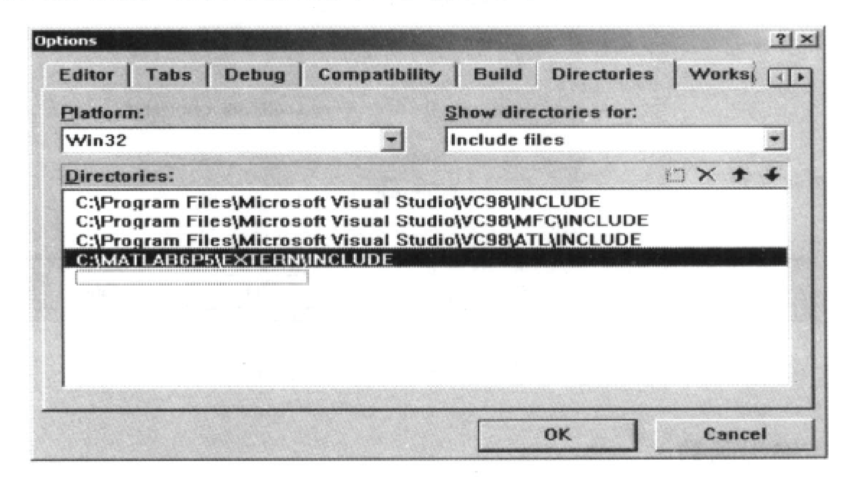


图 4-5　MEX 用到的头文件路径设置

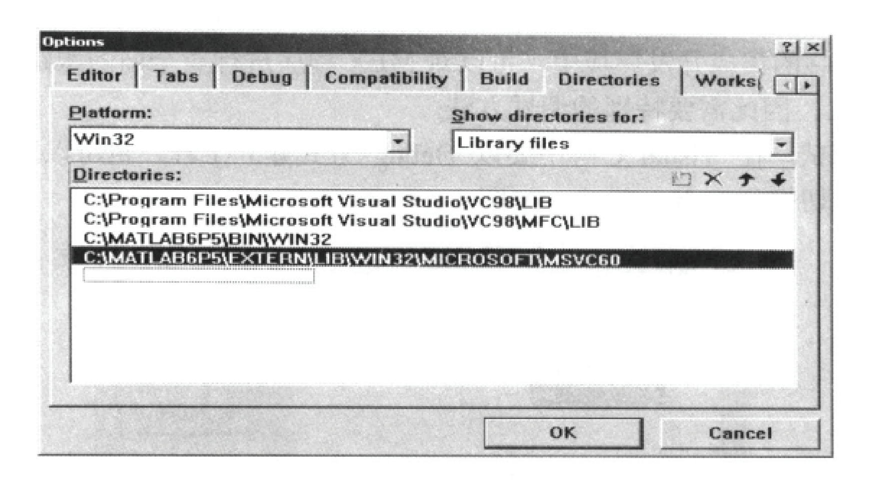


图 4-6　库文件路径设置对话框

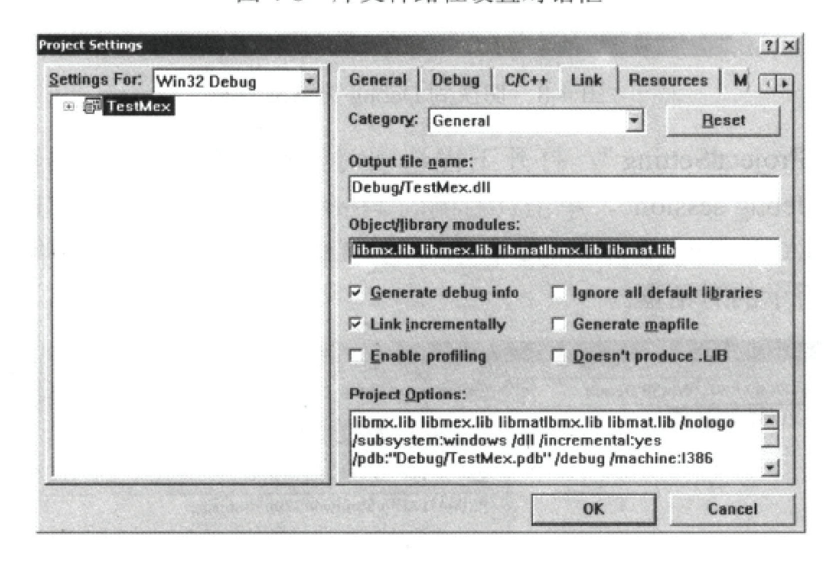


图 4-7　添加 MEX 文件可能用到的库文件

4.6.1.5　添加应用程序代码

完成以上设置后，MEX 文件就可以正常编译和链接了。但还没有添加含有 MEX 功能的代码，因此用户需要在必要的地方添加自己的代码，用 CPP 保存，并将文件加入工程中。

4.6.2　MEX 程序的调试

在不借助任何调试工具的条件下，调试 C 程序最简单的办法是使用 printf 命令。在 MATLAB C-MEX 中可以使用 mexPrintf，当然也可以使用 printf。但 mexPrintf 函数并不中断 MEX 程序的运行，如果程序出错，可能会给整个程序的运行带来致命的打击，尤其是在有内存管理等操作的程序中，往往会因 MEX 程序的非法操作，造成整个 MATLAB 系统的崩溃。为了提高调试效率，可以加入若干测试代码，当发现错误时及时提出。

MEX 函数提供了 mexErrMsgTxt 函数，它与 mexPrintf 函数不同，它仅能打印字符串，而不具备将后继参数转换为字符串的能力。更重要的是，它可以立即中断当前 MEX 程序的运行，并返回 MATLAB 命令行窗口，同时释放 MEX 程序中先前通过 mxCalloc 和 mxCreate 函数申请的内存，保证了 MEX 程序的安全退出。另一个常用的字符串打印函数是 mexWarnMsgTxt，它给出了警告信息，但不会退出 MEX 程序。

在 Visual C++的集成开发环境中，可以对 MEX 程序进行更全面的跟踪调试。由于 MEX 程序是一个 DLL，因此需要特别的调试方法。

为了便于调试，在 Visual C++中要以 Debug 方式建立工程，也可以通过菜单“Project|Configurations”切换到 Debug 模式，如图 4-8 所示。

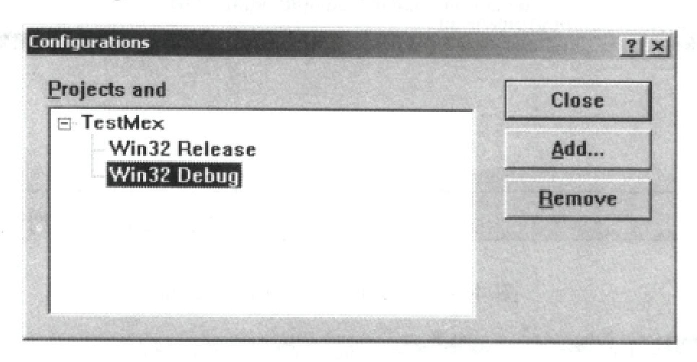


图 4-8　切换到 Debug 方式对话框

通过菜单“Project|Setting”，打开工程选项对话框，在 Debug 属性页，找到编辑框“Executable for debug session”，单击其右侧的按钮，出现浮动菜单，选择 Browse 选项，并找到位于 C:\MATLAB6p5\bin\win32 目录下的 MATLAB.exe，选中后单击“打开”按钮，则出现如图 4-9 所示的对话框。

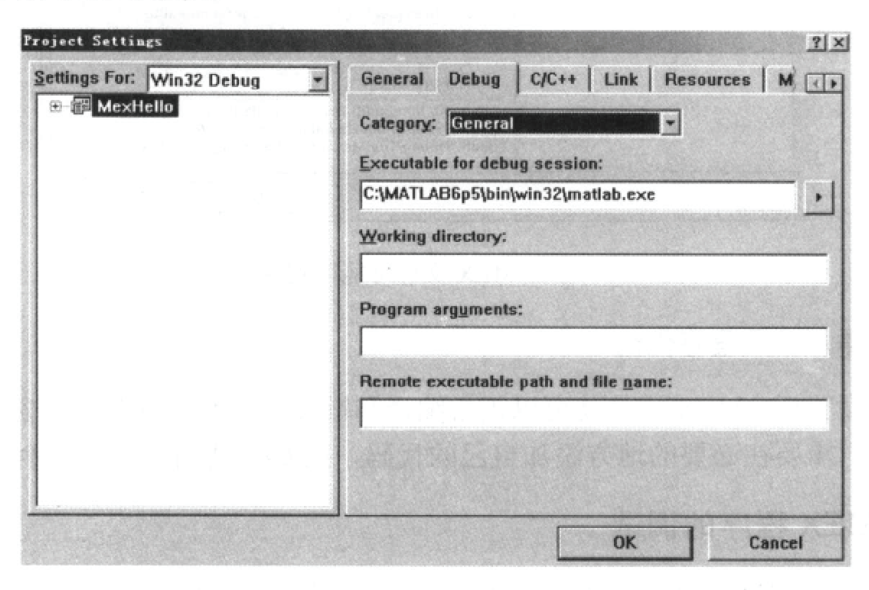


图 4-9　为调试 MEX 程序设置 MATLAB.exe 的路径

在 MEX 源程序中需要调试的合适位置设置断点。

按 F5 键运行程序就可以开始调试。此时会出现如图 4-10 所示的对话框。该对话框提示 MATLAB.exe 没有包含调试信息，不能进行调试。这是正常的，因为 MATLAB.exe 是以 release 方式发行的，不可能包含调试信息，事实上也无须对其进行调试，也无法通过 Visual C++调试它。因此，选中“Do not prompt in the future”前的复选按钮，单击“OK”按钮将其忽略即可。

MATLAB 启动后，默认的工作目录为当前工程的位置。由于 MEX 程序在当前工程的 Debug 目录下，因此为了运行 MEX 程序，还需要进入 Debug 目录：

>> cd debug

在程序断点处，可以运行 Visual C++提供的所有调试功能，如通过变量观察窗（Watch）看到最近处理的变量的数值。

调试完毕，可以在 MATLAB 窗口中输入 exit 命令，即可退出本次调试，也可以通过 Visual C++菜单中的“Build|Stop Debug”强行中断调试工作，它会自动关闭 MATLAB，Visual C++返回编辑状态。

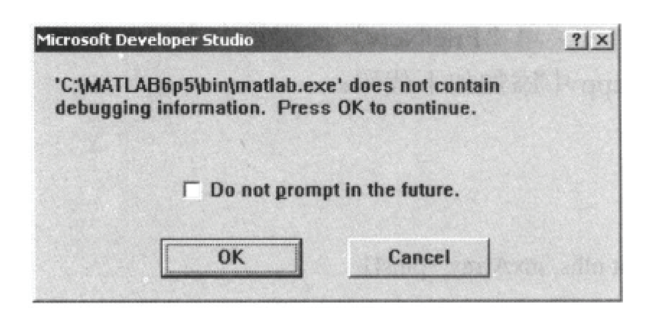


图 4-10　警告窗口

4.6.3　MEX 独立应用程序的发布

为了发布生成的 MEX 独立应用程序，需要生成一个文件包，包含这样一些类型文件：应用程序可执行文件、应用程序使用的所有定制的 MEX 文件、所有的 MATLAB 运行库。

4.7　MEX 编程实例

本节通过三个实例说明如何借助 Visual C++来编辑和调试 MEX 文件。第一个例子是一个简单的“Hello World”程序。第二个例子应用 MATLAB 自带的 yprime.c，第三个例子是自行编程实现稍复杂一些的功能。

【例 4-1】 一个简单的“Hello World”程序。

启动 Visual C++，选择菜单“File|New”，在 Project 页面中选择 MFC AppWizard(DLL)，建立一个名称为“MexHello”的工程。在 MFC AppWizard 的 Step 1 of 1 中选择“Regular DLL with MFC statically linked”，即选择静态链接的 MFC 类库。接下来，还需要完成以下几个步骤：

进行必要的环境设置，包括添加头文件和库文件的路径，以及添加必要的库文件。

修改 MexHello.def。可以在 Source Files 中看到 AppWizard 自动生成了以下几个文件：MexHello.cpp、MexHello.def、MexHello.rc 和 StdAfx.h 等。

需要在 MexHello.def 中指定该 DLL 文件的输出函数，也就是 mexFunction。为此，在 Visual C++的 Workspace 栏中，单击“FileView”属性页，展开“SourceFiles”栏，打开 MexHello.def 文件并编辑其内容，即在 Exports 后面加入 mexFunction 一行。编辑后的 MexHello.def 文件内容如下：

```
; MexHello.def : Declares the module parameters for the DLL.

LIBRARY        "MexHello"
DESCRIPTION    'MexHello Windows Dynamic Link Library'
```

```
EXPORTS
    ; Explicit exports can go here

;自行添加的代码
mexFunction
```

添加必要的代码。通过菜单“File|New”向工程中添加一个 C++ Source File，并将其命名为 main.cpp。在 main.cpp 中添加如下代码：

```
#include "stdafx.h"
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
  mexPrintf("Hello World\n");
//printf("Hello World\n");
//MessageBox(NULL,"Hello World","MEX file", MB_OK);

}
```

这时就可以编译和运行 MexHello 了，编译和链接之后，将在 debug 目录下得到 MexHello.dll。启动 MATLAB，进入 MexHello.dll 所在的目录，然后输入 MexHello，就可以看到“Hello World”出现在命令窗口。

当然，也可以按图 4-9 为 MEX 文件设置 MATLAB.exe 的路径。此时编译后运行，将自启动 MATLAB，且默认的目录为 MexHello 工程的目录。这时，只需在 MATLAB 环境中键入以下命令，就会看到“Hello World”出现在命令窗口。

```
>> cd debug
>> mexhello
Hello World
```

还可以在 Visual C++工程中使用 MFC 提供的函数进行调试，例如将 main.cpp 中 mexFunction( )里面的 mexPrint( )函数改为 MessageBox( )，即将 mexFunction( )的前两行代码注释掉，而去掉第三行的注释，然后编译运行。

注意：编译前要先关闭 MATLAB，否则会出现不能修改 MexHello.dll 的错误，因为 MexHello.dll 刚刚在 MATLAB 环境中运行。运行时，将会出现如图 4-11 所示的窗口。

【例 4-2】本例通过自行编写 C-MEX 代码，生成一个可以在 MATLAB 下运行的 C-MEX 文件。它计算素数（prime number，即除了 1 和它本身外，不能整除其他的数）的个数。需要经过以下几个步骤：

（1）启动 Visual C++，选择 File 菜单的 New 子菜单，在 Project 页面中选择建立一个“Win32 Dynamic-Link Library”的工程，工程名为“MexFunction”。在向导 Wizard 第一步，选择“A DLL that exports some symbols”。点击“结束”将建立 MexFunction 工程。

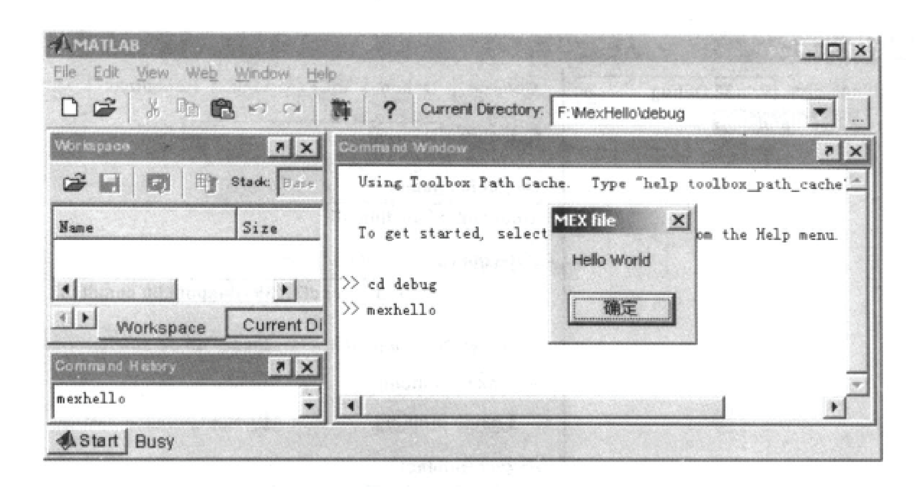


图 4-11 调用 MessageBox( )的 MEX 程序

（2）在菜单 Project 中选择 Setting 子菜单，选择 C/C++选项卡，在 Category 中选择 General，在 Preprocessor definitions 中添加 MATLAB_MEX_FILE。

（3）点击“File |New”菜单，选择生成一个 Text File 文件，如图 4-12 所示，并命名为 MexFunction.def。它定义输出的符号。

```
; testmexfunction.def : Declares the module parameters for the DLL.

LIBRARY         "ComputePrimes"
DESCRIPTION     'ComputePrimes Windows Dynamic Link Library'

EXPORTS
     ; Explicit exports can go here
  mexFunction
```

（4）在 Project 中选择 Setting 子菜单，选择 Link 选项卡，在 Category 中选择 General，如图 4-13 所示，在 Project Options 中添加如下的编译器选项：

/def: ".\mexFunction.def"

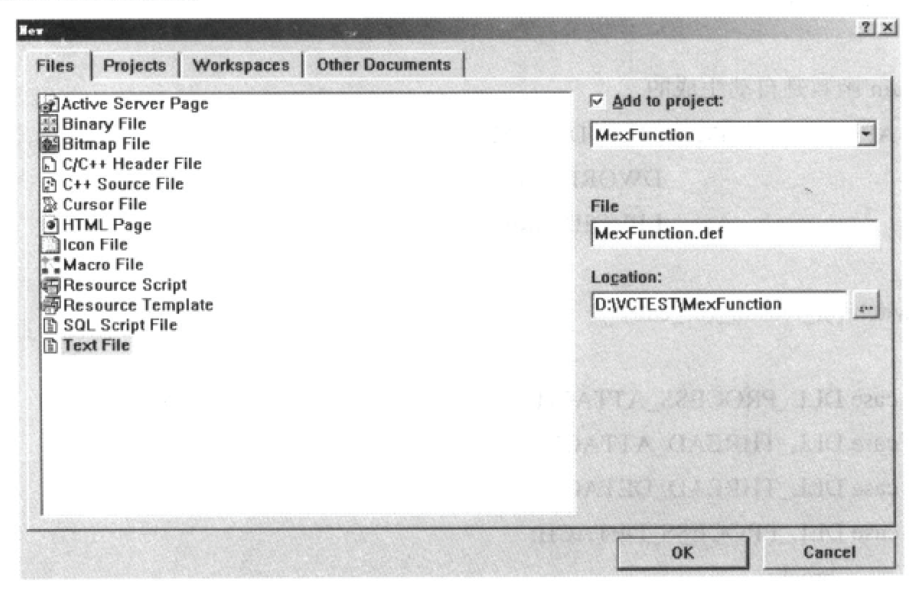


图 4-12 选择生成一个 Text File 文件

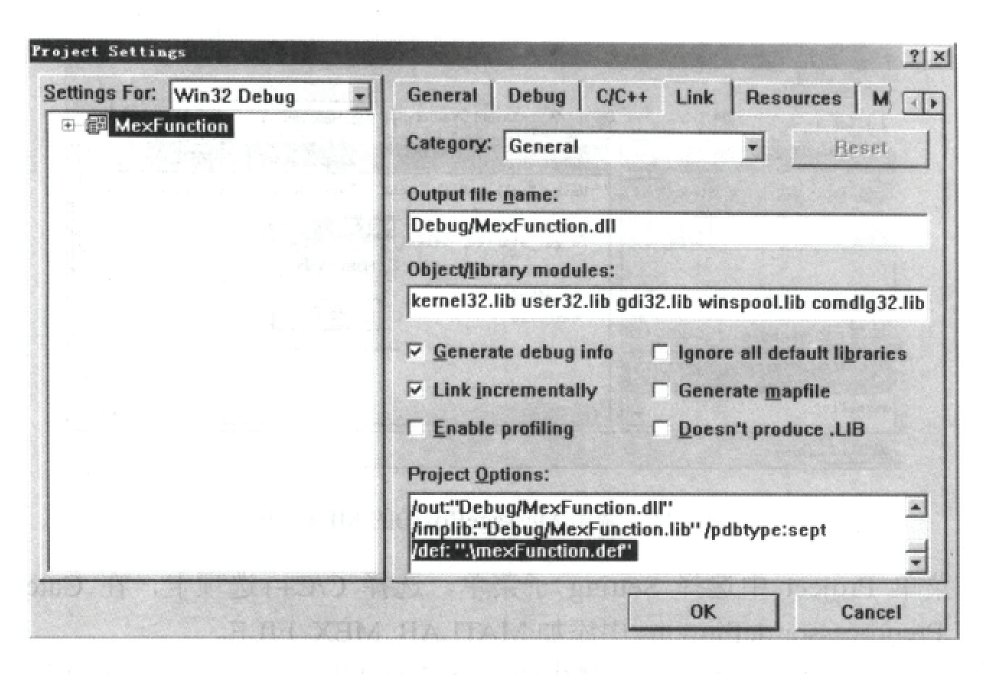


图 4-13　工程设置对话框

（5）在生成的 mexFunction.cpp 文件中添加以下头文件。

注意：后面的 4 条语句实际上是向工程中添加 4 个库文件，与第 4.1.6.1 节中添加库文件的效果一样。当然也可以按第 4.1.6.1 节进行设置，则不需要添加后面 4 条语句。

```
#include "MATLAB.h"
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmex.lib")
#pragma comment(lib, "libmatlb.lib")
```

（6）添加计算素数的代码。添加代码后，mexFunction 代码如下：

```
void ComputePrimes(double* y, int n);
BOOL IsPrime(int n);

//DllMain 函数是自动生成的
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD   ul_reason_for_call,
                       LPVOID lpReserved )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
```

```
    return TRUE;
}

void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[])
{
  // 输入/输出参数语法检查
  if (nrhs != 1)
  {
        mexErrMsgTxt("One input required.");
  }
  else if (nlhs > 1)
  {
        mexErrMsgTxt("Too many output arguments");
  }

  /* 输入参数必须不是复数，为此进行以下语法检查*/
  int mrows, ncols;
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);

  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
        !(mrows == 1 && ncols == 1))
  {
        mexErrMsgTxt("Input must be a noncomplex scalar integer.");
  }

  //读入输入变量
  double x, *y;
  x = mxGetScalar(prhs[0]);

  /* Create matrix for the return argument. */
  plhs[0] = mxCreateDoubleMatrix(mrows /* 1 */, (int) x, mxREAL);

  y = mxGetPr(plhs[0]);

  //计算素数
  ComputePrimes(y, (int) x);
}

//子程序，判断是否素数
void ComputePrimes(double* y, int n)
{
```

```
    int index=0, i=2;
    while (index!=n)
    {
        if (IsPrime(i))
        {
            y[index]=i;
            index++;
        }

        i++;
    }
}

//如果是素数，进行输出
BOOL IsPrime(int n)
{
    for (int i=2; i<=n/2; i++)
    {
        if (n%i==0)
            return FALSE;
    }
    return TRUE;
}
```

（7）编译将生成 MEX 文件 comprimes.dll，它可以在 MATLAB 下运行。在 MATLAB 提示符下输入：

```
>> computeprimes(8)
ans =
     2     3     5     7    11    13    17    19
```

当然，可以在 MATLAB 中编辑一个同名的 MATLAB 文件，即 computeprimes.m。它只包含一些帮助信息。下面的代码就是一个例子。这样，用户在使用此函数时输入"help computeprimes"，即可得到关于本函数用法的帮助信息。

```
function y=ComputePrimes(n)
%This program calculates first n prime number
%and save them to a matrix (y)
%size of matrix is 1xn
%
%example:
%y=ComputePrimes(8)
%
%y =
%
%     2     3     5     7    11    13    17    19
```

4.8 小　　结

MEX 文件虽然具有较强大的功能，但并不是对所有的应用都恰当。MATLAB 是一个高效率的编程系统，特别适合于工程计算、系统仿真等应用。它的最大优点就是将人们从复杂的程序编写中解放出来。因此，能够用 M 文件完成的程序应尽量使用 MATLAB 编写，除非遇到必须使用 MEX 文件的情况。

需要说明的是，MATLAB 到 C 语言程序的转换可以由两种途径完成，其一是 MATLAB 自己提供的 C 语言翻译程序 mcc，另一种是原第 3 方公司 MathTools 开发的 Matcom。后者出现较早，功能远比 MATLAB 自己的翻译程序强大，所以 MathTools 公司已经被 MathWorks 公司收购，并已将其开发技术融入了 MATLAB 6.5（Release 13）以后的版本中。

从程序转换的速度来看，因为 mcc 和 Matcom 都沿用了 MATLAB 的程序运算机制，所以不可能大幅度地提高程序运行速度。相反地，如果将 MATLAB 程序中明显的瓶颈部分用 C 语言按照 MEX 格式编写，则可以大大加快速度。

本章详细介绍 C-MEX 文件，包括 C 语言的 MEX 函数库以及 MEX 文件在 MATLAB 和 Visual C++环境中的调试。

第 5 章　通过 MATLAB 引擎实现混合编程

5.1　MATLAB 引擎简介

Windows 是一个多任务的操作系统，允许多个应用程序同时运行，应用程序之间有多种机制可以互相通信。事实上，MATLAB 可以被 C、Fortran 等程序调用，也可以调用 C、Fortran 程序。从编程复杂度的角度看，通过引擎实现混合编程是最简单的途径，即在其他应用程序中向 MATLAB 发送命令，控制它的运行。在这种情况下，用户的 C、Fortran 应用程序将 MATLAB 作为一个计算或图形显示的引擎来调用。

所谓 MATLAB 引擎（Engine），是指一组 MATLAB 提供的接口函数，它支持 C 和 Fortran 两种语言。通过这些接口函数，用户可以在 C 或 Fortran 的应用程序中实现对 MATLAB 的控制，MATLAB 引擎函数库在后台工作。

具体地，Visual C++之类的通用编程平台一般利用 MATLAB 引擎完成以下功能：

- 调用一个数学函数或者子程序来处理数据。这时，MATLAB 就是一个强有力、编程灵活且高效的数学函数库。
- 调用 MATLAB 的图形函数库，将一些 Visual C++编程复杂度高、难以实现的图形显示任务由 MATLAB 处理。这时，MATLAB 就是一个功能强大的图形函数库。
- 利用 MATLAB 编程高效和强大的计算能力，结合 Visual C++之类的通用编程平台输入、输出功能强大，编写图形界面简单的特点，可以为特定的任务建立一个高效的、界面友好的系统，达到缩短开发周期、降低开发难度的目的。

通过 MATLAB 引擎，应用程序会打开一个新的 MATLAB 进程。可以控制它完成任何计算和绘图工作，对所有的数据结构都提供支持。因此，实际上应用程序是预先存储了 MATLAB 的命令，然后自动地送给 MATLAB 执行。

与其他各种编程接口相比，MATLAB 引擎方式最简单也最全面，而且工作时不需要 MATLAB 整个与应用程序相连，只需要一小部分引擎通信函数库与程序相连。因此，它大大节约了系统资源，还可以充分利用网络资源，将计算任务繁重的工作放到性能最好、运算能力最强的计算机上，通过网络运行 MATLAB 引擎。

5.2　MATLAB 引擎库函数

MATLAB 引擎库包含了一系列从外部应用程序调用和控制 MATLAB 引擎的函数。表 5-1 列举了 C 语言的引擎库函数，这些函数都使用了 eng 前缀名。Fortran 语言的引擎函数库与 C 语言的引擎函数库类似，本书从略。

C 语言的引擎库函数的定义都在头文件 engine.h 中定义。需要说明的是，后 4 个函数 engSetVisible、engGetVisible、engGetVariable 和 engPutVariable 是 MATLAB 6.1 新增加的。而 engPutArray 和 engGetArray 已经过时，它们分别被 engPutVariable 和 engGetVariable 所取代，

但仍可以使用。

表 5-1　C 语言引擎函数库

引擎函数	功能描述
engOpen	启动 MATLAB 计算引擎
engGetArray	从 MATLAB 引擎获得一个 MATLAB 矩阵，用于数据交换
engPutArray	从应用程序向 MATLAB 引擎发送一个 MATLAB 矩阵，用于数据交换
engEvalString	执行一个 MATLAB 命令
engOutputBuffer	创建字符缓冲区，以获取 MATLAB 文本输出
engOpenSingleUse	打开一个单独的非共享的 MATLAB 计算引擎
engClose	关闭 MATLAB 引擎
engSetVisible	设置 MATLAB 的显示或隐藏状态
engGetVisible	返回 MATLAB 显示状态
engGetVariable	从 MATLAB 工作区（Workspace）获取一个变量
engPutVariable	将指定名称的 MATLAB 变量存入 MATLAB 工作区

Engine.h 中对这几个函数的定义如下：

```
/*
 * @(#)engine.h      generated by: makeheader 3.10   Sat May 18 22:03:04 2002
 *
 *          built from: ../include/copyright.h
 *                      engapi.c
 *                      engapiv4.c
 *                      engapiv5.c
 *                      fengapi.c
 *                      fengapiv5.c
 *                      modver/modver.c
 */
#ifndef engine_h
#define engine_h

/* $Revision: 1.4 $ */
/*
 * Copyright (c) 1985-2001 The MathWorks, Inc.
 * All Rights Reserved.
 */

#ifdef __cplusplus
    extern "C" {
#endif
```

```
#include <stdio.h>
#include "matrix.h"

typedef struct engine Engine;  /* Incomplete definition for Engine */

/*
 * Execute matlab statement
 */
extern int engEvalString(
  Engine      *ep,          /* engine pointer */
  const char *string        /* string for matlab t execute */
  );

/*
 * Start matlab process for single use.
 * Not currently supported on UNIX.
 */
extern Engine *engOpenSingleUse(
                const char *startcmd, /* exec command string used to start matlab */
                void *reserved, /* reserved for future use, must be NULL */
                int *retstatus /* return status */
);

/*
 * SetVisible, do nothing since this function is only for NT
 */
extern int engSetVisible(
           Engine *ep,            /* engine pointer */
           bool newVal
           );

/*
 * GetVisible, do nothing since this function is only for NT
 */
extern int engGetVisible(
           Engine *ep,            /* engine pointer */
           bool* bVal
           );

/*
 * Start matlab process
 */
```

```
extern Engine *engOpen(
  const char *startcmd /* exec command string used to start matlab */
  );


/*
 * Close down matlab server
 */
extern int engClose(
  Engine      *ep              /* engine pointer */
  );

/*
 * Get a variable with the specified name from MATLAB's workspace
 */
extern mxArray *engGetVariable(
  Engine      *ep,        /* engine pointer */
  const char *name/* name of variable to get */
  );

/*
 * Put a variable into MATLAB's workspace with the specified name
 */
extern int engPutVariable(
             Engine *ep,             /* engine pointer */
             const char *var_name,
             const mxArray *ap      /* array pointer */
             );

/*
 * register a buffer to hold matlab text output
 */
extern int engOutputBuffer(
  Engine      *ep,         /* engine pointer */
  char  *buffer,     /* character array to hold output */
  int         buflen          /* buffer array length */
  );

#define engOpenV4( )          cannot_call_engOpenV4
#define engGetFull( )        engGetFull_is_obsolete
#define engPutFull( )        engPutFull_is_obsolete
#define engGetMatrix( )      engGetMatrix_is_obsolete
```

```
#define engPutMatrix( )     engPutMatrix_is_obsolete

#if defined(V5_COMPAT)
#define engPutArray(ep, ap)     engPutVariable(ep, mxGetName(ap), ap)
#define engGetArray(ep, name) engGetVariable(ep, name)
#else
#define engPutArray( ) engPutArray_is_obsolete
#define engGetArray( ) engGetArray_is_obsolete
#endif /* defined(V5_COMPAT) */

#ifdef __cplusplus
    } /* extern "C" */
#endif

#endif /* engine_h */
```

下面从功能描述、函数原型和参数说明三个方面对 engine.h 中定义的 C 语言引擎函数进行详细介绍。

1. engOpen

功能描述：启动 MATLAB 计算引擎。

函数原型：extern Engine *engOpen(const char *startcmd)。

参数说明：输入参数为一个字符指针，函数通过该指针指向的字符串所包含的命令与 MATLAB 建立一个连接，打开一个 MATLAB 进程，返回一个 MATLAB 引擎类型的指针。

在 Windows 系统中，该函数打开一个 ActiveX 通道并将它与 MATLAB 进行连接，函数所启动的 MATLAB 是在安装时就已经注册了的。如果所安装的 MATLAB 未注册，可以使用如下命令进行注册：matlab /regserver。

2. engGetArray

功能描述：从 MATLAB 引擎的工作区获得一个 MATLAB 矩阵。

函数原型：mxArray *engGetArray(Engine *ep, const char *name)。

参数说明：参数 ep 是引擎指针，name 是希望从 MATLAB 引擎的工作区间获得的矩阵的名字。返回值是一个 mxArray 结构体的指针。如果函数执行成功，则返回一个指向新分配的 mxArray 结构体的指针，否则返回 0。

注意：最后要释放该函数新创建的 mxArray 结构体。

3. engPutArray

功能描述：从应用程序向 MATLAB 引擎发送一个 MATLAB 矩阵。

函数原型：int engPutArray(Engine *ep, const mxArray *mp)。

参数说明：参数 ep 为 MATLAB 引擎类型的指针，mp 为指向矩阵的指针，指向需要输出到 MATLAB 计算引擎工作区中的 mxArray 结构体。如果 MATLAB 引擎的工作不包含所指

定的 mxArray 结构体，则函数自动创建一个名称为 mp 的新矩阵；如果引擎的工作包含所指定的 mxArray 结构体，则函数将用新值覆盖旧值。若函数执行成功则返回 0，否则返回 1。

4. engEvalString

功能描述：执行一个 MATLAB 命令。

函数原型：extern int engEvalString(Engine *ep, const char *string)。

参数说明：ep 为 engOpen 打开的 engine 指针。String 为字符串，指向一个需要在 MATLAB 计算引擎的工作区中执行的字符串。函数返回 0 表示成功执行，返回 1 则表示 ep 对应的 MATLAB Engine 已经关闭。

注意：需要在 MATLAB 计算引擎的工作区中执行的字符串要符合 MATLAB 的语法。

5. engOutputBuffer

功能描述：创建字符缓冲区，以获取 MATLAB 文本输出。

函数原型：extern int engOutputBuffer(Engine *ep, char *buffer, int buflen)。

参数说明：ep 为事先打开的 MATLAB Engine 指针，buffer 为字符型缓冲区的指针，buflen 为最大保存的个数，通常也就是缓冲区的大小。该函数为 ep 指向的 MATLAB 引擎设置从 buffer 指向的长度为 buflen 的缓冲区。只有设置了输出缓冲区后，所有后继的 engEvalString 操作产生的输出才会保存在缓冲区中。

6. engOpenSingleUse

功能描述：打开一个单独的非共享的 MATLAB 计算引擎（UNIX 系统不支持）。

函数原型：extern Engine *engOpenSingleUse(const char *startcmd, void *reserved, int *retstatus)。

参数说明：startcmd 为命令字符串以启动 MATLAB 进程，reserved 为保留字，而 retstatus 为返回的状态。函数的返回值为打开的计算引擎。这个函数不常用。

7. engClose

功能描述：关闭 MATLAB 引擎。

函数原型：extern int engClose(Engine *ep)。

参数说明：输入参数 ep 为已经打开的 MATLAB 计算引擎。返回“0”表示成功关闭，返回“1”表示返回出错。

8. engGetVariable

功能描述：从 MATLAB 工作区（Workspace）获取一个变量。

函数原型：extern mxArray *engGetVariable(Engine *ep, const char *name)。

参数说明：MATLAB 6.5 新增加的引擎函数，替代函数 engGetArray，使用方法与 engGetArray 类似。其中，ep 为指向 MATLAB 引擎的指针，name 为一个字符常量，它是 MATLAB 引擎工作区存在的一个变量的名称。

9. engPutVariable

功能描述：将指定名称的 MATLAB 变量存入 MATLAB 工作区。

函数原型：extern int engPutVariable(Engine *ep, const char *var_name, const mxArray *ap)。

参数说明：ep 为指向 MATLAB 引擎的指针；ap 为指向即将输出到 MATLAB 引擎工作区中的 mxArray 类型的变量名；name 为指向 mxArray 类型的指针，它是在 MATLAB 引擎区中的变量名。

10. engSetVisible

功能描述：设置 MATLAB 的显示或隐藏状态（对 Windows 2000/NT 系统有效）。

函数原型：extern int engSetVisible（Engine *ep, bool newVal）。

参数说明：ep 是 MATLAB 的引擎指针，当 newVal 为 1 时，显示打开 MATLAB 引擎即 MATLAB 在 Windows 任务栏是可见的。反之，当 newVal 为 0 时，隐藏 MATLAB 窗口，但 MATLAB 仍然在后台运行。

11. engGetVisible

功能描述：判断当前的 MATLAB 窗口是否是打开的。

函数原型：extern int engGetVisible（Engine *ep, bool* bVal）。

参数说明：ep 是指向 MATLAB 引擎的指针，bVal 是指向返回值的指针。因为应用程序中很少需要查询 MATLAB 窗口是否打开，故这个函数也很少用。

5.3 Visual C++调用 MATLAB 引擎时的环境设置

当在 Visual C++环境下调用 MATLAB 引擎时，编译接口环境的配置较为简单，主要有以下两步（假设 MATLAB 安装在目录 C:\MATLAB6p5 下）。

1. 添加 MATLAB 引擎库的头文件和库函数的路径

打开菜单“Tools”，选取“Options... ->Directories”。在“Show directories for”选项卡选取“Include files”，添加“C:\MATLAB6P5\EXTERN\INCLUDE”（如图 5-1 所示）。它是 MATLAB 引擎库的头文件 engine.h 所在的目录。然后在“Show directories for”选项卡中选取“Library Files”，添加“C:\MATLAB6P5\EXTERN\LIB\WIN32\MICROSOFT\MSVisual C++60”（如图 5-2 所示），因为这些引擎库用到的动态链接库都在此目录下。

2. 完成 MATLAB 引擎对应的静态链接库的导入工作

在菜单“Project->Setting”下，选取“Link”选项卡，在“Object/Library Modules”里添加 libmx.lib、libmex.lib 和 libeng.lib。注意三个文件名中间以空格分开（如图 5-3 所示）。

注意：库函数 libeng.lib, libmat.lib, libmex.lib 和 libmx.lib 在 MATLAB 6.0 以前版本中没有，需要通过命令行操作得以生成这几个静态链接库。以 libeng.lib 为例。

进入 Visual C++的安装目录后，再进入\bin 子目录，然后按以下格式运行命令 lib:

```
lib /def: c:\matlab6p5\extern\include\libmx.def /machine:ix86 /out: libmx.lib/LOGOGO
```

在 C++Builder 中，MATLAB 提供的 def 文件允许用户通过 implib 命令生成相应的 lib 文件。

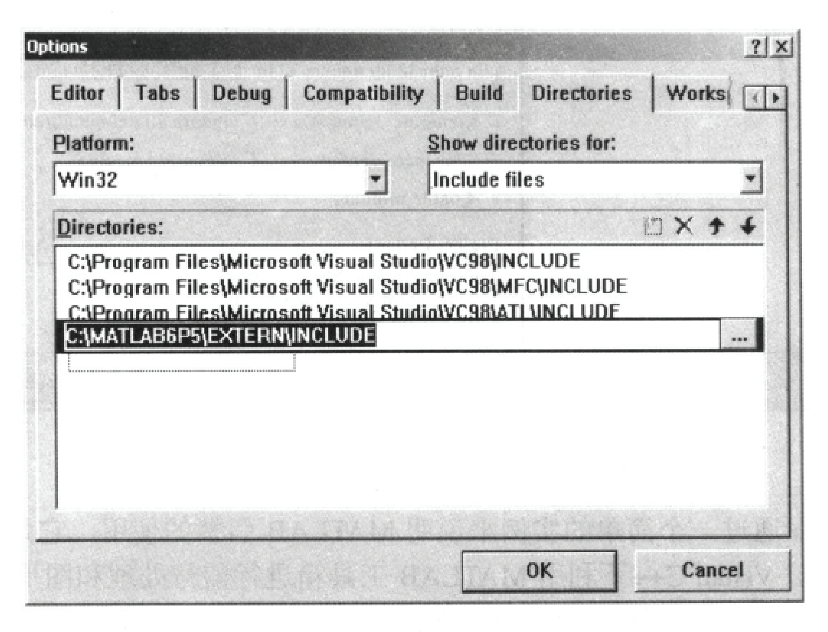


图 5-1 Include files 路径配置对话框

经试验发现，第二步的设置可以改为：在 Visual C++ 6.0 环境中，选择“Project”菜单中的“Add to Project”命令，然后选择 Files 选项，将 C:\matlab6p5\extern\include\lib\win32\Microsoft\msvc60 目录下的 libeng.lib、libmex.lib 和 libmx.lib 文件添加到工程中。它同样可以将三个静态链接库添加到当前工程中来。

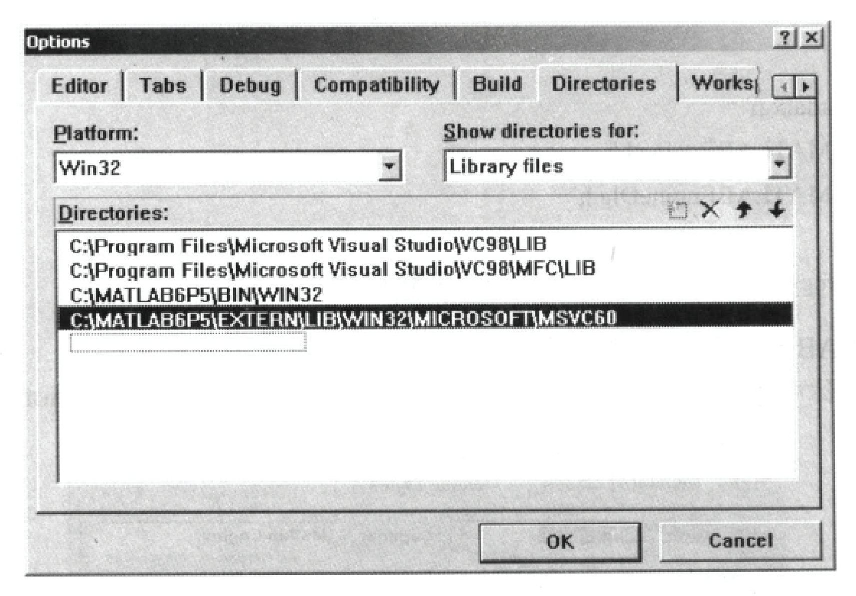


图 5-2 Library files 路径配置对话框

图 5-3　Engine 方式静态链接库设置

【例 5-1】下面通过一个简单的实例来说明 MATLAB 引擎的使用，它是一个基于对话框的 MFC 应用程序，Visual C++下利用 MATLAB 工具箱进行信号处理和图形显示的操作。

启动 Visual C++，建立一个新工程，选择 MFC AppWizard(exe)，工程名为 MATLABEngine。在“Step 1 of 1”中选择“Dialog based”，即生成一个基于对话框的 MFC 应用程序。其余各项采用默认设置。

使用库的头文件 matlib.h。首先添加 MATLAB 引擎库的头文件和库函数的路径，然后在 MATLABEngineDlg.cpp 中添加 engine.h，如下：

```
// MATLABEngineDlg.cpp : implementation file
//

#include "stdafx.h"
#include " MATLABEngine.h"
#include " MATLABEngineDlg.h"

#include "engine.h"          //添加 MATLAB 引擎头文件
```

添加 MATLAB 引擎库对应的静态链接库文件。

在对话框窗体中添加一个按钮，如图 5-4 所示设置按钮的标题（Caption）和控件 ID。

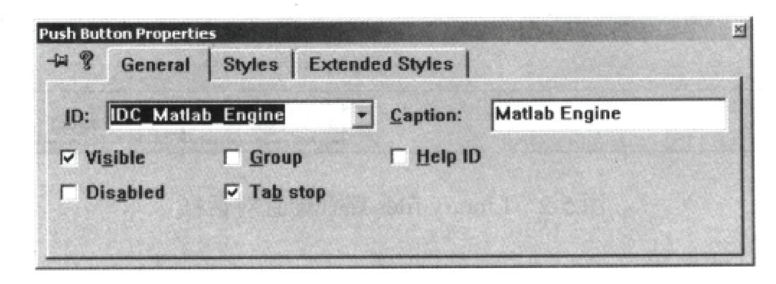


图 5-4　按钮控件的标题和控件 ID 设置对话框

通过 Visual C++的 ClassWizard，添加按钮的单击响应事件，并添加如下代码。

```
void CMATLABEngineDlg::OnMATLABEngine( )
{
  Engine *ep;     //定义 MATLAB 引擎变量

  MessageBox("单击确定按钮，Windows 正在准备启动 MATLAB 引擎!",
       "MATLAB 引擎 ", MB_OK | MB_ICONINFORMATION);

  if (!(ep=engOpen("\0")))  //打开 MATLAB 引擎
  {
       fprintf(stderr, "\n MATLAB 引擎启动失败!\n");
       MessageBox("MATLAB 引擎启动失败!","MATLAB 引擎 ",
                 MB_OK |MB_ICONERROR);
       exit(-1);
  }

  MessageBox("单击确定按钮，隐藏 MATLAB!","MATLAB 引擎",
           MB_OK | MB_ICONINFORMATION);
  engSetVisible(ep, 0);     //隐藏 MATLAB 窗口

  MessageBox("单击确定按钮，系统重新显示 MATLAB!",
           "MATLAB 引擎",  MB_OK | MB_ICONINFORMATION);
  engSetVisible(ep, 1);     //重新显示 MATLAB 窗口

  mxArray *T = NULL;
  double time[10] = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 };

  //生成矩阵
  T = mxCreateDoubleMatrix(1, 10, mxREAL);
  memcpy((void *)mxGetPr(T), (void *)time, sizeof(time));

  //将矩阵变量写入 MATLAB 引擎的工作区
  engPutVariable(ep,"T", T);

  //通过 MATLAB 引擎执行 MATLAB 命令
  engEvalString(ep, "D = .5.*(-9.8).*T.^2;");
  engEvalString(ep, "plot(T,D);");
  engEvalString(ep, "title('Position vs. Time for a falling object');");
  engEvalString(ep, "xlabel('Time (seconds)');");
  engEvalString(ep, "ylabel('Position (meters)');");
  engEvalString(ep, "grid on;");
```

```
    //释放矩阵变量 T 占用的内存空间
    mxDestroyArray(T);

    MessageBox("通过例程，说明 MATLAB 的图形显示能力!",
               "MATLAB 引擎",  MB_OK | MB_ICONINFORMATION);

    //画一朵花
    engEvalString(ep, "x=-8:0.5:8;");
    engEvalString(ep, "y=x;");
    engEvalString(ep, "[Y,X]=meshgrid(y,x);");
    engEvalString(ep, "R=sqrt(X.^2+Y.^2)+eps;");
    engEvalString(ep, "Z=2*sin(R)./R;");
    engEvalString(ep, "surf(X,Y,Z);");

    MessageBox("关闭 MATLAB 引擎，系统将退出 MATLAB 应用程序!",
          "MATLAB 引擎",  MB_OK | MB_ICONINFORMATION);
    engClose(ep);  //关闭 MATLAB 引擎，退出 MATLAB
}
```

按 F7 键编译后再按 Ctrl+F5 组合键运行，将会出现对话框。单击“MATLAB Engine”按钮，将会出现 Windows 的标准信息框，出现“单击确定按钮，Windows 正在准备启动 MATLAB 引擎!”提示信息。单击“确定”按钮，可以看到任务栏出现 MATLAB 的图标，它表示 MATLAB 已经启动。然后会出现“隐藏 MATLAB”的信息提示框，同样单击“确定”按钮，可以看到任务栏上的 MATLAB 图标立即隐藏（注意：MATLAB 此时仍在后台运行，只不过在任务栏隐藏了它的图标）。然后，按照提示进行操作，系统会在任务栏出现“重新显示 MATLAB 的图标”信息，并先后出现如图 5-5 和图 5-6 所示的实验结果。

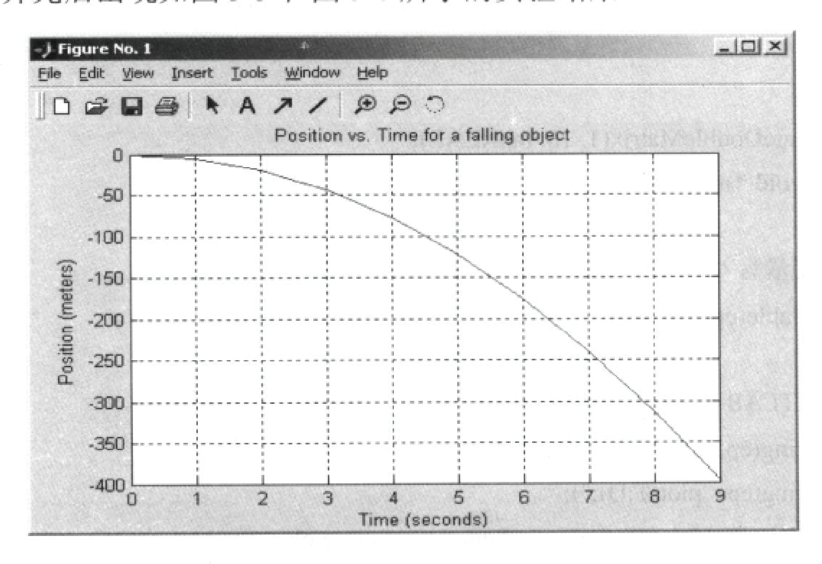


图 5-5　实验图形一

可以看出，MATLAB 引擎工作方式实质上是 Visual C++等通过编程平台实现了和 MATLAB 的交互，它将需要在 MATLAB 环境中执行的命令预先“存储”在 Visual C++的应

用程序中，然后自动地送给 MATLAB 执行。当然，可以通过向 MATLAB 引擎的工作空间读取或写入变量实现 MATLAB 和 Visual C++的数据交互。

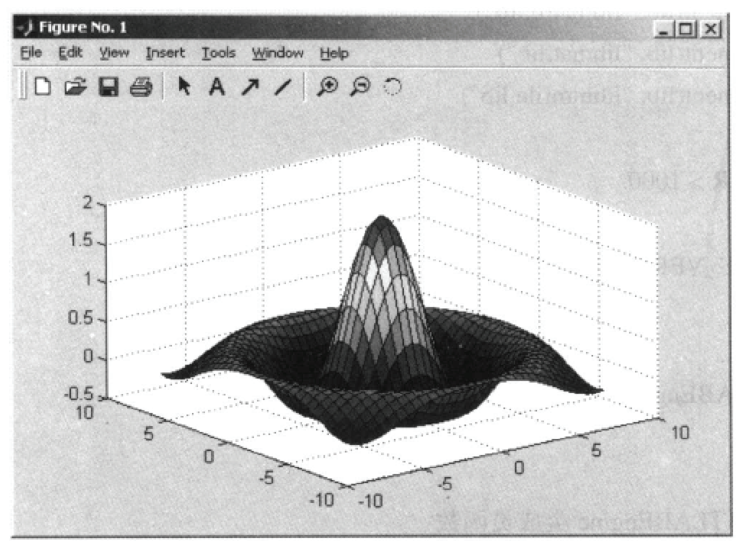


图 5-6 实验图形二

5.4 MATLAB 引擎类的封装

5.4.1 CMATLABEng 类的定义和实现代码

通过 MATLAB 引擎方式实现混合编程较简单，主要是添加必要的头文件和库文件，然后调用 MATLAB 引擎库函数。尽管如此，为了进一步简化 MATLAB 引擎的使用，A. Riazi 博士封装了 MATLAB 的引擎类。

以下是 MATLAB 引擎类 CMATLABEng 的头文件 MATLABEng.h：

```
// MATLABEng.h
#ifndef _MATLAB_ENGINE_H_
#define _MATLAB_ENGINE_H_

//////////////////////////////////////////////////////////////////////////////


#ifndef FALSE
#define FALSE    0
#endif

#ifndef TRUE
#define TRUE     1
#endif

#include " MATLAB.h"
#include "Engine.h"

//MATLAB library – 自动链接 MATLAB 的库函数
```

```
#pragma comment(lib, "libeng.lib")
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libmatlb.lib")
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmmfile.lib")

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CMATLABEng
{
public:
  //定义的 MATLABEngine 类成员函数
  int OutputBuffer(char *p, int n);
  void OpenSingleUse(const char *startcmd, void *dcom, int *retstatus);
  int GetVisible(bool* value);
  int SetVisible(bool value);
  mxArray* GetVariable(const char* name);
  int PutVariable(const char *name, const mxArray *mp);
  int EvalString(const char* string);
  void Open(const char* StartCmd);
  int Close( );

  CMATLABEng( );    //构造函数
  virtual ~CMATLABEng( );      //析构函数

protected:
  Engine* pEng;
};

#endif // _MATLAB_ENGINE_H_
```

以下是 MATLAB 引擎类 CMATLABEng 的实现文件 MATLABEng.cpp 代码:

```
// MATLABEngine.cpp: implementation of the CMATLABEng class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include " MATLABEng.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
```

```
////////////////////////////////////////////////////////////

CMATLABEng::CMATLABEng( )
{
  pEng=NULL;
}

CMATLABEng::~CMATLABEng( )
{
  if (pEng!=NULL)
        Close( );
}

void CMATLABEng::Open(const char *StartCmd)
{
  pEng=engOpen(StartCmd);
}

int CMATLABEng::Close( )
{
  int Result=engClose(pEng);
  if (Result==0)    //Success
        pEng=NULL;

  return Result;
}

int CMATLABEng::EvalString(const char *string)
{
  return (engEvalString(pEng, string));
}

mxArray* CMATLABEng::GetVariable(const char *name)
{
  return (engGetVariable(pEng, name));
}


int CMATLABEng::GetVisible(bool* value)
{
  return (engGetVisible(pEng, value));
}

void CMATLABEng::OpenSingleUse(const char *startcmd, void *dcom, int *retstatus)
```

```
{
  pEng=engOpenSingleUse(startcmd, dcom, retstatus);
}

int CMATLABEng::OutputBuffer(char *p, int n)
{
  return (engOutputBuffer(pEng, p, n));
}

int CMATLABEng::PutVariable(const char *name, const mxArray *mp)
{
  return (engPutVariable(pEng, name, mp));
}

int CMATLABEng::SetVisible(bool value)
{
  return (engSetVisible(pEng, value));
}
```

5.4.2 CMATLABEng 说明

封装 CMATLABEng 类是为了方便 MATLABEngine 的使用。CMATLABEng 类共有 9 个成员函数，每一个成员函数在 MATLAB 引擎 API 中都有对应的函数，且类似。建议与第 1.2 节中 MATLAB 引擎库函数对照阅读。下面对 CMATLABEng 类的成员函数的原型、功能描述和参数进行说明。

1. Open

函数原型：void Open(const char* StartCmd)。
功能描述：打开 MATLAB 引擎进程，使 MATLAB 作为计算引擎。
参数说明：StartCmd 为指向字符串的指针。在 Windows 系统下，一般设置为 NULL。

2. OutputBuffer

函数原型：int OutputBuffer(char *p, int n)。
功能描述：指定 MATLAB 输出的缓冲。
参数说明：p 为指向字符串缓冲的指针；n 为字符串缓冲的长度。

这个函数指定一个字符串缓冲，以便 engEvalString 返回一般在屏幕上显示的输出。EvalString( ) 一般会丢弃它执行的命令产生的标准输出，而 OutputBuffer(ep,p,n)通知后继的 EvalString 保存指针 p 指向的缓冲区中输出的前 n 个字符。关闭输出函数，可以通过 OutputBuffer(ep,NULL,0)写成。

3. OpenSingleUse

函数原型：void OpenSingleUse (const char *startcmd, void *dcom, int *retstatus)。

功能描述：打开一个单独的、非共享的 MATLAB 进程。

参数说明：startcmd 同 Open(const char* StartCmd)相同。在 Windows 系统下，一般设置为 NULL。Dcom 为保留字，一般为 NULL。retstatus 返回状态及各种引起失败的原因。

这个函数允许启动多个 MATLAB 进程，以让 MATLAB 作为计算引擎。它返回独一无二的 MATLAB 引擎标识符，如果失败则返回 NULL。

4. GetVisible

函数原型：int GetVisible (bool* value)。

功能描述：返回 MATLAB 引擎的状态在任务栏是显示还是隐藏。

参数说明：value 为布尔型。

5. SetVisible

函数原型：int SetVisible (bool value)。

功能描述：显示或隐藏 MATLAB 引擎在任务栏的图标。

参数说明：value 为布尔型。若为 1 则显示 MATLAB 在任务栏的图标，若为 0 则关闭 MATLAB 在任务栏的图标。

6. GetVariable

函数原型：mxArray* GetVariable(const char* name)。

功能描述：从 MATLAB 引擎的工作区中拷贝一个变量。

参数说明：name 为指向字符串常量的指针。它实际上就是 MATLAB 引擎工作区中 mxArray 类型的变量名。当变量名不存在时，返回 NULL。

注意：当使用完变量后，要释放内存空间。

7. PutVariable

函数原型：int PutVariable(const char *name, const mxArray *mp)。

功能描述：输出一个变量到 MATLAB 引擎工作区。

参数说明：name 为向 MATLAB 引擎工作区中输出的变量名；mp 为指向 mxArray 类型的指针。当成功时返回值为 1，当失败时返回值为 0。

8. EvalString

函数原型：int EvalString(const char* string)。

功能描述：向 MATLAB 引擎传送一个 MATLAB 命令并执行。

参数说明：String 为字符串，指向一个需要在 MATLAB 计算引擎的工作区中执行的字符串。函数返回 0 表示成功执行，返回 1 则表示 ep 对应的 MATLABEngine 已经关闭了。

注意：在 MATLAB 计算引擎的工作区中执行的字符串要符合 MATLAB 的语法。

9. Close

函数原型：int Close( )。

功能描述：关闭 MATLAB 引擎。

参数说明：返回 0 表示成功关闭，返回 1 表示返回出错。

5.4.3 CMATLABEng 说明和使用方法

为了通过 CMATLABEng 类使用 MATLAB 引擎功能，必须在 Visual C++工程进行以下编译环境设置：

（1）添加 matlab.h 到 stdafx.h 或者需要的接口和执行文件中。

（2）添加库文件到 Visual C++工程中。根据具体情况，可能需要添加的库文件有 libeng.lib, libmx.lib, libmatlb.lib, libmat.lib 和 libmmfile.lib。

（3）添加 CMATLABEng 类的定义文件 MATLABEng.h 和实现文件 MATLABEng.cpp 到工程中。

（4）定义一个 CMATLABEng 类型的变量。

（5）通过 CMATLABEng 类的成员函数控制 MATLAB 引擎，如发送数据到 MATLAB 引擎和从 MATLAB 引擎接收数据以及绘图和数值计算等。

（6）编译工程并执行。

5.5 应 用 实 例

【例 5-2】本例将例 5-1 的功能改为由 CMATLABEng 类实现。为了简化问题的说明，将例 5-1 基于对话框的应用程序改为基于 Win32 Console Application（控制台应用程序）。具体包含以下步骤：

启动 Visual C++，建立一个新工程，选择 Win32 Console Applications，工程名为 MATLABEngClass。在“Step 1 of 1”选择“A simple application”，即生成一个简单的 DOS 控制台应用程序。其余各项采用默认设置。

使用库的头文件 matlib.h。首先按 1.3 节中“Visual C++调用 MATLAB 引擎时的环境设置”的第一步，添加 MATLAB 引擎库的头文件和库函数的路径，然后在 stdafx.h 中添加以下代码：

```
#include <stdio.h>
#include <conio.h>
#include <matlab.h>
```

添加 CMATLABEng 类的定义文件 MATLABEng.h 和实现文件 MATLABEng.cpp 到工程中。创建空的 C++ Source File 和 C/C++ Header File，分别命名为 MATLABEng.cpp 和 MATLABEng.h，然后将 1.4.1 节中的代码复制到其中，也可以将光盘中的这两个文件复制到当前工程目录中，并通过菜单“Project”选择“Add to Project”添加到工程中。

在自动生成的 MATLABEngClass.cpp 中添加以下代码：

```
// MATLABEngClass.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include " MATLABEng.h"
```

```
#define message(x) printf(x"\n\n")

int main(int argc, char* argv[])
{
  CMATLABEng matlab;

  //open new matlab session
  message("启动 MATLAB 引擎，请稍候...");
  matlab.Open(NULL);

  message("准备在任务栏隐藏 MATLAB 图标");
  matlab.SetVisible(FALSE);
  message("单击任意键继续");
  getch( );

  message("在任务栏显示 MATLAB 图标");
  matlab.SetVisible(TRUE);
  message("单击任意键继续");
  getch( );

  mxArray *T = NULL;
  double time[10] = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 };

  //生成矩阵
  T = mxCreateDoubleMatrix(1, 10, mxREAL);
  memcpy((void *)mxGetPr(T), (void *)time, sizeof(time));

  //发送矩阵到 MATLAB 引擎
  message("Send matrix T to matlab");
  matlab.PutVariable("T", T);

  //通过 MATLAB 引擎执行 MATLAB 命令
  matlab.EvalString("D = .5.*(-9.8).*T.^2;");

  //绘图
  message("Plot(T, D)");
  matlab.EvalString("plot(T,D);");
  matlab.EvalString("title('Position vs. Time for a falling object');");
  matlab.EvalString("xlabel('Time (seconds)');");
  matlab.EvalString("ylabel('Position (meters)');");
  matlab.EvalString("grid;");
```

```
    //pause to see results
    message("Press any key to continue");
    getch( );

    //释放矩阵T所点的内存空间
    mxDestroyArray(T);

    //显示MATLAB的图形输出能力
    message("显示MATLAB的图形输出能力!");
    matlab.EvalString("x=-8:0.5:8;");
    matlab.EvalString("y=x;");
    matlab.EvalString("[Y,X]=meshgrid(y,x);");
    matlab.EvalString("R=sqrt(X.^2+Y.^2)+eps;");
    matlab.EvalString("Z=2*sin(R)./R;");
    matlab.EvalString("surf(X,Y,Z);");

    //等待输入任意键，以观察图形输出结果
    getch( );

    //关闭MATLAB引擎
    matlab.Close( );

    return 0;
}
```

按F7键进行编译后，按Ctrl+F5组合键运行，会出现如图5-7所示的DOS控制台程序窗口。按提示进行操作，任务栏上的MATLAB图标会显示后隐藏再显示，并会出现如图5-8和图5-9所示的实验结果。

可以看出图5-8和图5-5相同，图5-9与图5-6相同，即直接通过MATLAB引擎库函数和封装后的CMATLABEng类取得了相同的实验结果，但通过封装后的CMATLABEng类使用更简单，更符合基于MFC的习惯。

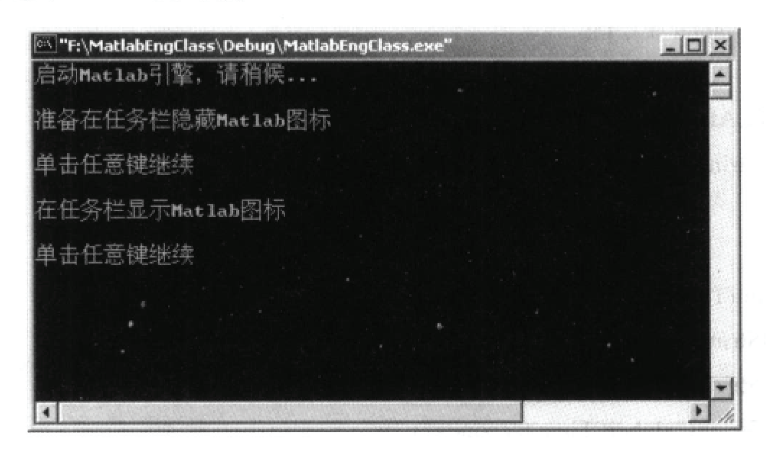


图5-7　控制台程序窗口

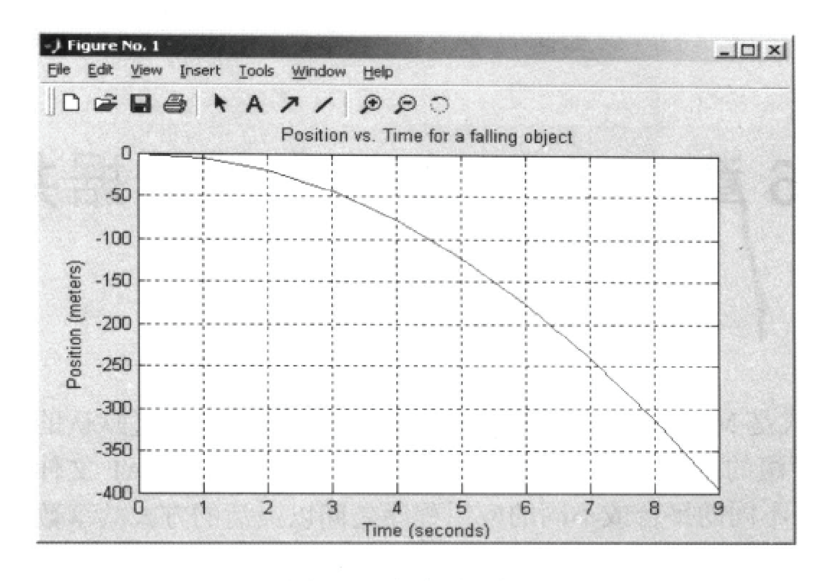


图 5-8　实验图形之一

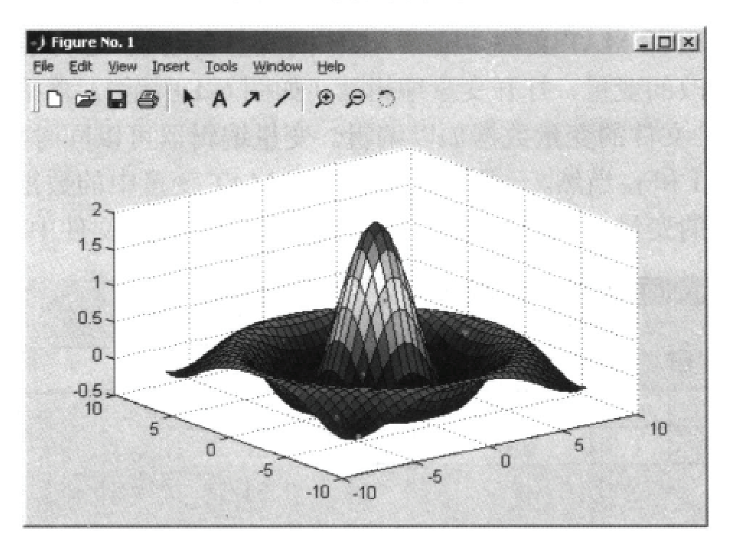


图 5-9　实验图形之二

5.6 小　　结

本章详细介绍了 MATLAB 的 Engine 应用程序接口，包括直接使用 MATLAB 提供的 Engine 库 API 函数以及将这些 API 函数封装为 CMATLABEng 类的方法，并举例说明了在 Visual C++程序中通过 MATLAB 引擎库 API 函数以及 CMATLABEng 类控制 MATLAB 的运行，它们的运行效果相同。相对来说，CMATLABEng 类较 MATLAB 引擎库 API 更易使用。

第 6 章　MAT 文件实现数据共享

6.1　MAT 文件简介

MAT 文件格式是 MATLAB 专用的数据存储的标准格式，也是默认的文件格式。它的文件名是以.mat 为后缀的。MAT 文件把数据存储为二进制格式。MAT 文件提供了一种简便的机制，允许在两个不同的平台或不同的应用程序之间以灵活的方式共享数据。

MAT 文件是 MATLAB 专用的二进制数据文件，因此不能用文本编辑器打开，只能在 MATLAB 中打开。修改 MAT 文件中变量的方法是启动 MATLAB 后，在菜单 File 中选择 Open，然后找到要打开的 MAT 文件，则 MAT 中的变量自动载入到 MATLAB 的工作区中，这时可以双击需要修改的变量，打开变量编辑器（如图 6-1 所示），直接对变量的内容进行修改，对工作区中 MAT 文件的变量数据加以编辑。变量编辑器可以同时打开多个变量，图 6-1 中就同时打开了变量 i 和 j。当然，一般很少手工修改 MAT 变量中的数据，而是通过 MATLAB 的命令修改工作区间的变量，然后保存工作区间的变量到 MAT 文件中。

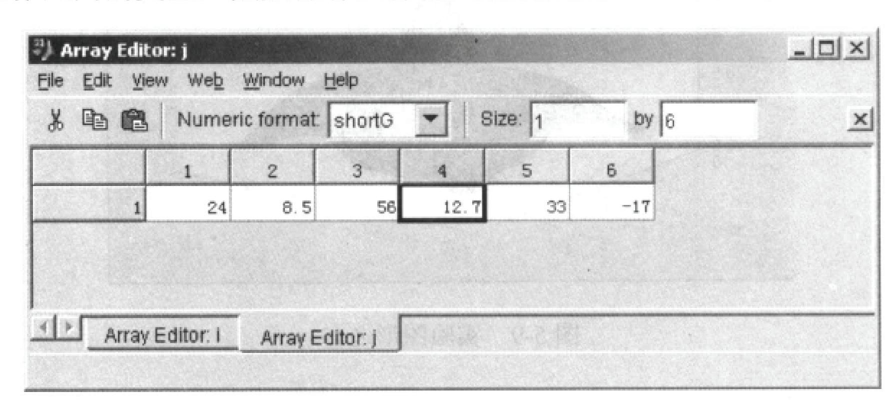


图 6-1　MATLAB 中变量编辑器

MATLAB 的 save 命令可以将 MATLAB 系统内部数据保存为 MAT 文件，而 load 命令可以将磁盘上的 MAT 文件读入到 MATLAB 系统中。除此之外，为了有效地管理 MAT 文件，以及在 MATLAB 外部读取和创建 MAT 文件，MATLAB 提供了一个子程序库，可以在 C 或 Fortran 程序中直接调用这些子程序来创建和读取 MAT 文件。

6.2　操作 MAT 文件

6.2.1　MAT 文件格式

MAT 数据格式是 MATLAB 数据存储的标准格式。与常见的 BMP 格式的图像文件等一样，MAT 文件也有自己的存储格式。MAT 文件由 128 字节的 MAT 文件头和尾随在后的数据单元组成，每个数据单元头部都有一个 8 字节的标志，这个标志表示在这个数据单元里有多

少数据，以及以什么方式读/写数据。一般的读/写方式有 16 位数据、32 位数据、浮点或别的形式。MAT 文件的结构如图 6-2 所示。

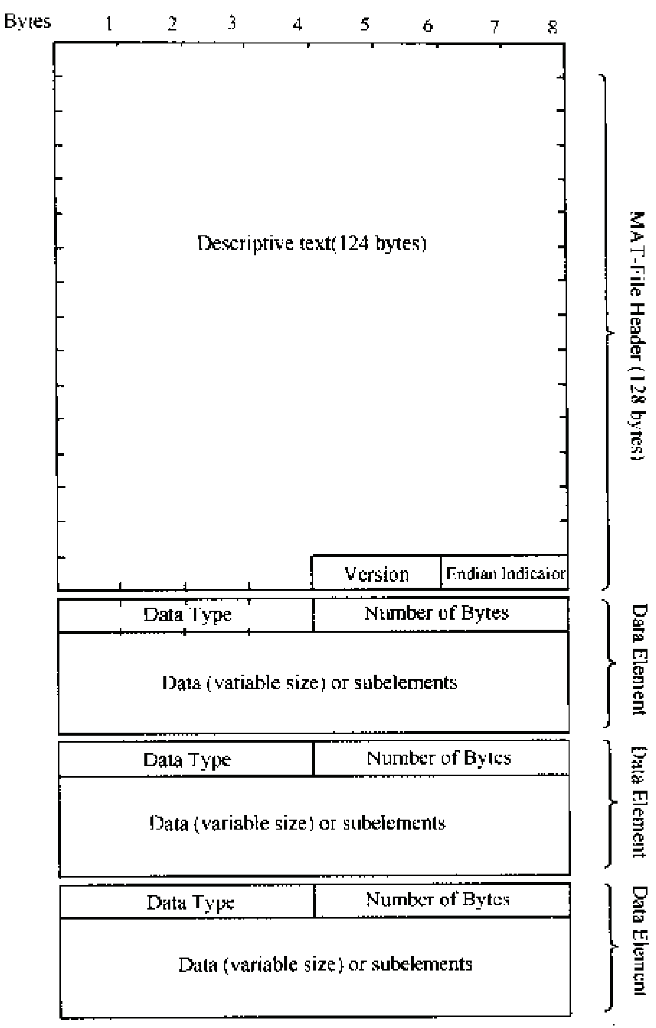


图 6-2 MAT 文件格式

从图 6-2 中可以看出，MAT 文件前面有 124 字节的描述信息，它们都是可读的文本信息。这些文件信息描述了 MATLAB MAT 文件的创建信息和数据信息。MAT 文件头包括以下信息：

- MATLAB 版本；
- MAT 文件创建的平台；
- 数据和文件被创建的时间。

在 Windows 系统中，阅读这些信息的方法是在 MATLAB 环境中使用 type 命令：

```
>> type my_matfile.mat
```

在 Windows 系统中，显示的信息如下：

MATLAB 5.0 MAT-file, Platform: PCWIN, Created on: Fri Aug 08 09:04:36 2003。

它表明 MAT 文件是在 Windows 平台下 MATLAB 5.0 版本格式的 MAT 文件。

128 字节的 MAT 文件头后面的 4 字节是 MAT 文件的标志域，包含用来创建 MAT 文件的 MATLAB 软件的版本等信息。紧跟着 MAT 文件头后的是数据单元。每个数据单元由 8 字节的标志开始，接着是此单元中的数据元素。8 字节的标志包含两部分信息：数据类型和字节数。数据类型也就是单元中每个数据的大小和格式，MAT 文件支持许多种格式的数据，其中包含有符号的和无符号的 8 位、16 位、32 位和 64 位数据，还有一些表示 MATLAB 矩阵的特殊数据类型。字节数是一个 32 位的整数，表示数据单元里有多少字节，不包含数据单元开始的 8 字节标志。

详细的 MAT 文件格式可参阅 MATLAB 随机文档或从 MathWorks 公司下载。

6.2.2 操作 MAT 文件的 MATLAB API

一般情况下，并不需要了解 MAT 文件的内部具体格式。为了简化在 MATLAB 环境之外对 MAT 文件的使用，MATLAB 给出了一系列 MATLAB API 来完成 MAT 文件的读取与存储，而 MATLAB API 完全屏蔽了 MAT 文件格式。当然，当需要在一个 MATLAB API 不支持的系统中读/写 MAT 文件时，用户必须了解 MAT 文件的存储格式，以便开发 MATLAB MAT 文件的读/写子程序。MATLAB API 支持 MATLAB 可以运行的平台，因此在 Windows 系统下，完全可以运用 MATLAB API 来完成对 MAT 文件的读/写，而屏蔽 MAT 文件的内部格式。MathWorks 强烈建议用户采用 MAT 文件 API 来读/写 MAT 文件，而不是自己读/写 MAT 文件的代码，因为当 MATLAB 升级时，MAT 版本改变意味着用户必须重写读/写 MAT 文件的代码。

因此，先了解一下与读/写 MAT 文件有关的 API。在 Windows 系统下，与 MAT 文件有关的文件有需要包含的头文件和库文件。头文件主要有 matrix.h 和 mat.h。其中，matrix.h 包含了 MATLAB 中基本的数据类型、矩阵的定义和操作方法，mat.h 包含了 MAT 文件的创建、读/写函数的定义。库文件是与平台有关的。有 Windows 系统下，库文件有 libmat.lib 和 libmx.lib。表 6-1 列出了有关 MAT 文件的子目录。

表 6-1 MAT 文件的子目录

相关文件	目录
include 文件	<MATLAB>\extern\include
库文件	<MATLAB>\bin\win32
MATLAB 所附例程	<MATLAB>\extern\examples\eng_mat

6.2.2.1 mat.h 中的 API 函数

在操作 MAT 文件的子程序库中，它们都定义在 mat.h 中。头文件 mat.h 中定义的主要函数有 13 个，其函数名称和功能描述见表 6-2。

表 6-2 mat.h 中定义的 MAT 文件头

函数名称	功能描述
matOpen	打开 MAT 文件
matclose	关闭 MAT 文件
matGetDir	取得 MAT 文件的变量列表
matGetArray	从 MAT 文件中取一个矩阵变量

续表

函数名称	功能描述
matPutArray	向 MAT 文件中存一个矩阵变量
matDeleteArray	从 MAT 文件中删除一个矩阵变量
matGetFp	取 MAT 文件中的 C 语言 FILE 句柄
matGetNextArray	从 MAT 文件中取下一个矩阵变量
matPutArrayAsGlobal	向 MAT 文件中存一个矩阵，使得当用 load 命令装入这个 MAT 文件时，该矩阵对应的变量成为 global 变量
matGetArrayHeader	读取 MAT 文件中的 MATLAB 矩阵头信息
matGetNextArrayHeader	读取 MAT 文件中下一个 MATLAB 的矩阵头信息

下面对 mat.h 中定义的 MAT API 进行详细的介绍。

1. matOpen( )

功能描述：打开一个 MAT 文件。

函数原型：MATFile *matOpen(const char *filename, const char *mode)。

参数说明：filename 为一个指向字符串的指针，包含了希望打开的 MAT 文件名；mode 为字符指针，用来说明打开文件的模式，有以下几种取值：

①“r”表示用只读方式打开文件。

②“w”表示用只写方式打开文件。如果当前目录不存在 filename 指向的文件，则创建新的；如果存在 filename 指向的文件，则会覆盖原来的文件。

③“u”表示用读/写方式打开文件。

④“w4”表示创建一个 MATLAB 4.x 版本的 MAT 文件。

函数打开一个 MAT 文件用于读/写，如果执行成功则返回 MAT 文件类型指针，否则返回空指针。

2. matClose( )

功能描述：关闭一个 MAT 文件。

函数原型：int matClose(MATFile *mfp)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针。如果函数执行成功，返回值为 0，否则返回一个写文件错误。

3. matDeleteVariable ( )

功能描述：从 MAT 文件中删除一个矩阵。

函数原型：int matDeleteVariable(MATFile *mfp, const char *name)。

参数说明：mfp 指向一个 MAT 文件的 MATFile 类型指针；name 为指向一个希望删除的矩阵名的字符指针。如果函数执行成功，返回值为 0，否则返回一个非零值。

4. matGetDir ( )

功能描述：获得 MAT 文件中所有矩阵的目录。

函数原型：char **matGetDir(MATFile *mfp, int *num)。

参数说明：mfp 指向一个 MAT 文件的 MATFile 类型指针；num 为文件的数目。如果函数执行成功，将返回一个字符指针数组，数组每个元素均为字符指针，指向的字符串表示 MAT 文件中矩阵的目录。如果函数执行失败，则返回参数 num 为-1，并且返回一个空指针。如果 num 为 0，则表示 MAT 文件中没有矩阵。

注意：该函数通过 mxCalloc 分配内存，在程序结束时，必须使用函数 mxFree 释放内存，否则会导致内存泄漏。

5. matGetFp( )

功能描述：获得 MAT 文件的 C 语言文件句柄。通过该句柄，用户可以方便地使用 C 语言的库函数 feof 或 ferror 来判断出错的原因。

函数原型：FILE *matGetFp(MATFile *mfp)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针。

6. matGetNextVariable( )

功能描述：获得 MAT 文件中下一个矩阵的数据。

函数原型：mxArray *matGetNextVariable(MATFile *mfp, const char *name)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针；name 为包含 mxArray 变量的地址。函数返回值是一个指向新分配的 mxArray 结构的指针。当到达文件的末尾或者发生错误时，返回值为空，可以通过标准的 C 库函数来决定错误原因。在程序结束时，必须使用函数 mxFree 释放内存，否则会导致内存泄漏。

MATLAB 5.3 版本中的 matGetNextArray ( )已过时并被本函数代替，但 matGetNexArray( )函数仍可以使用。

7. matGetNextVariableInfo( )

功能描述：获取 MAT 文件下一个变量的信息。

函数原型：mxArray *matGetVariableInfo(MATFile *mfp, const char *name)。

参数说明：mfp 为指向下一个 MAT 文件的 MATFile 类型指针，函数读取 mfp 指向的 MAT 文件的下一个矩阵的信息并返回一个 mxArray 类型的指针。该函数必须紧接在 matOpen 之后，也就是在该函数与 matOpen 之间不能有其他文件操作，否则该函数无法进行定位。在程序结束时，需要将该函数分配的 mxArray 结构体删除，否则会导致内存泄漏。如果到达文件的结尾或者函数执行错误，则返回值为 0。

MATLAB 5.3 版本中的 matGetNextArrayHeader ( )已过时并被本函数代替，但 matGetNex ArrayHeader( )函数仍可以使用。

8. matGetVariable( )

功能描述：从 MAT 文件中获取一个矩阵变量。

函数原型：mxArray *matGetVariable(MATFile *mfp, const char *name)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针；name 为指向一个希望获得的矩阵名的字符指针。如果函数执行成功，函数将创建一个 mxArray 结构体对象，并将获得

的矩阵数据拷贝到新分配的 mxArray 结构体对象中，返回指向该对象的指针，否则返回为 NULL。在程序结束之前必须将函数产生的 mxArray 结构体对象删除。

9. matGetVariableInfo( )

功能描述：从指定的 MAT 文件中读取指定名称的变量的头信息。

函数原型：mxArray *matGetVariableInfo(MATFile *mfp, const char *name)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针；name 为 mxArray 类型的变量名。它只获得矩阵的头信息，这些信息是除了矩阵的 pr、pr、ir 和 jc 外的所有信息。需要注意的是，这里读取的信息是只读的，不能回写或保存到 MAT 文件中。如果函数执行成功，那么在返回的 mxArray 结构体中，pr、pi、ir 和 jc 均被设置为-1。

10. matPutVariable( )

功能描述：将矩阵变量内容写入 MAT 文件中。

函数原型：int matPutVariable(MATFile *mfp, const char *name, const mxArray *mp)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针；name 为 mxArray 类型的变量名。此函数可以将一个 mp 指向的 mxArray 结构体写入到 mfp 所指向的 MAT 文件中。如果文件中存在同名的 mxArray 结构体，那么将覆盖原来的值；如果存在同名的 mxArray 结构体，那么将此结构体添加到文件的末尾。

MATLAB 5.3 版本中的 matPutArray( )已过时并被本函数代替，但 matPutArray ( )函数仍可以使用。

11. matPutVariableAsGlobal( )

功能描述：将矩阵的内容写入到 MAT 文件中。

函数原型：int matPutVariableAsGlobal(MATFile *mfp, const char *name, const mxArray *mp)。

参数说明：mfp 为指向一个 MAT 文件的 MATFile 类型指针，mp 为一个指向 mxArray 结构体的指针。这个函数可以将一个 mp 指向的 mxArray 结构体写到 mfp 所指向的 MAT 文件中。如果文件中已存在同名的 mxArray 结构体，那么将覆盖原来的值；如果不存在同名的 mxArray 结构体，那么将此结构添加到文件的末尾。函数执行成功返回 0，否则返回一个非零值。

此函数与 matPutVariable 功能类似，唯一的区别在于读取矩阵时。MATLAB 在读取用 matPutVariableAsGlobal 存放的矩阵时，会将矩阵放到 MATLAB 的全局工作空间，而 matPutVariable 无此功能。

以前版本中的 matPutArrayAsGlobal( )已过时并被本函数代替，但 matPutArrayAsGlobal ( )函数仍可以使用。

6.2.2.2 matrix.h 中的 API

mxArray 类型以及大量的以 mx 为前缀的函数都定义于 matrix.h 中。数据结构 mxArray 是一种很复杂的数据结构，与 MATLAB 中的 array 对应。因此，在单独使用 mxArray 及相关

函数的代码中，应该包含头文件 matrix.h。

下面介绍一些常见的函数，它们与矩阵类型 mxArray 操作密切相关。

1. 创建和清除数组

MATLAB 有多种变量类型，对应于每种类型，基本上都有一个函数用于创建它，不过它拥有相同的数据结构，即 mxArray。该数据结构很复杂，其中包含了数组的类型，也包含了实际的数据。

数组的建立一般采用 mxCreateXXX( )形式的函数。例如，新建一个二维的 double 类型的数组，应该用函数 mxCreateDoubleMatrix，其函数原型如下：

```
mxArray *mxCreateDoubleMatrix(int m, int n, mxComplexity flag)。
```

其中，m 和 n 为矩阵的行数和列数。参数 flag 为常数，用于区别矩阵中的数据是实数还是复数，这两个常数分别为 mxREAL 和 mxCOMPLEX。

类似的 mxCreateXXX( )函数见表 6-3，它们都同于创建不同类型数据的矩阵。

表 6-3 矩阵创建函数 mxCreateXXX( )

函 数	功能描述
mxCreateLogicalArray	创建一个 N 维的逻辑 mxArray 类型的矩阵，并初始化所有元素为 false
mxCreateLogicalScalar	用指定的值创建一个逻辑标量
mxCreateLogicalMatrix	创建一个二维的逻辑 mxArray 类型的矩阵，并初始化所有元素为 false
mxCreateDoubleScalar	创建一个双精度的阵列，所有的元素用指定的值初始化
mxCreateSparse	创建一个二维的稀疏矩阵
mxCreateSparseLogicalMatrix	创建一个二维的稀疏逻辑矩阵
mxCreateStringFromNChars	用指定的字符串创建一个 1×N 的字符串阵列
mxCreateString	按指定的字符串创建一个字符串阵列
mxCreateCharArray	创建一个字符阵列，并用指定的字符初始化
mxCreateCharMatrixFromStrings	按字符串的内容创建一个字符矩阵
mxCreateCellMatrix	创建一个二维的空阵列，每个元素都为 NULL
mxCreateCellArray	创建一个 N 维的空阵列，每个元素都为 NULL
mxCreateStructMatrix	用指定的字段创建一个二维的结构体阵列，所有的值都初始化为 NULL
mxCreateStructArray	用指定的字段创建一个 N 维的结构体阵列，所有的值都初始化为 NULL

用 mxDestroyArray( )函数从内存中删除一个数组，函数原型如下：

```
void mxDestroyArray(mxArray *array_ptr)
```

2. 管理数组的维数

获取数组每一维上元素的个数，可以用 mxGetM ( )和 mxGetN ( )函数。其中，mxGetM ( )获得数组第一维的元素个数。对于矩阵来说，也就是行数。其函数声明如下：

```
int mxGetM(const mxArray *array_ptr)
```

获得其他维的元素个数，可以用函数 mxGetN ( )，函数声明如下：

```
int mxGetN(const mxArray *array_ptr)
```

对于矩阵来说，该函数返回的是矩阵的列数。对于多维数组来说，它返回的是从第 2 维到最后一维的各维元素的乘积。

如果要得到某一特定维的元素的个数，则要使用 mxGetDimensions( )函数。它的函数声明如下：

```
const int *mxGetDimensions(const mx Array *array_ptr)
```

各维的元素个数将保存在一个 int 数组中返回。对于最常用的矩阵来说，只要用 mxGtM( )和 mxGetN( )两个函数就足够了。

另外，还可以通过 mxGetNumberOfDimensions( )获得数组的总维数。

设置矩阵的行数和列数，需要使用 mxSetM( )和 mxSetN( )两个函数。它们的原型分别为：

```
extern void mxSetM(mxArray *pa, int m )
extern void mxSetN(mxArray *pa, int n)
```

3、判断数组类型

在对 mxArray 类型的变量进行操作之前，可以验证一下其中数组的数据类型，比如是否为 double 数组、整数、字符串、逻辑值等，以及是否为某种结构、类，或者是某种特殊类型，比如是否为空数组，是否为 InF、NaN 等。常见的判断函数有：

```
extern bool mxIsChar(const mxArray *pa)
extern bool mxIsStruct(const mxArray *pa)
extern bool mxIsLogical(const mxArray *pa)
extern bool mxIsCell(const mxArray *pa)
extern bool mxIsNumeric(const mxArray *pa)
extern bool mxIsComplex(const mxArray *pa)
extern bool mxIsSparse(const mxArray *pa)
extern bool mxIsDouble(const mxArray *pa)
extern bool mxIsSingle(const mxArray *pa)
extern bool mxIsInt8(const mxArray *pa)
extern bool mxIsUint8(const mxArray *pa)
extern bool mxIsUint16(const mxArray *pa)
extern bool mxIsInt32(const mxArray *pa)
extern bool mxIsUint32(const mxArray *pa)
extern bool mxIsInt64(const mxArray *pa)
extern bool mxIsUint64(const mxArray *pa)
extern bool mxIsEmpty( const mxArray *pa)
extern bool mxIsInf(double x )
extern bool mxIsNaN(double x )
```

对于每个函数的功能，不再详细说明，从函数的原型就很容易看到。

4. 管理数组的数据

对于常用的 double 类型的数组，可以用 mxGetPr( )和 mxGetPi( )两个函数分别获得其实部和虚部的数据指针。这两个函数的声明如下：

```
extern void mxSetPr(
     mxArray *pa,        /* pointer to array */
     double   *pr            /* real data array pointer */
     );
extern double *mxGetPi(
     const mxArray *pa        /* pointer to array */
     );
```

这样就可以通过获得的指针对 mxArray 类型数组中的数据进行读/写操作。例如可以用函数 MAT API 函数 matGetArray( )从 MAT 文件中读入 mxArray 类型的数组，然后用 mxGetPr( )和 mxGetPi( )获得数据指针，并对其中的数据进行处理，最后调用函数将修改后的数据存入 MAT 文件中。

限于篇幅，本书只介绍了一些常用的 mx 函数。感兴趣的读者可以参考 MATLAB 应用程序接口手册。

6.3 Visual C++调用 MAT 时的环境设置

为了使用 MATLAB API 来操作 MAT 文件，必须进行必要的环境设置。与操作 MAT 数据文件有关的头文件主要有 matrix.h 和 mat.h。其中，matrix.h 包含了 MATLAB 中基本的数据类型、矩阵的定义和操作方法，mat.h 包含了 MAT 文件的创建、读/写等函数的定义。库文件是与平台有关的。在 Windows 系统下，库文件有 libmat.lib 和 libmx.lib。具体的设置步骤如下：

（1）添加头文件和库文件路径。在 Tools 中选择 Options，出现如图 6-3 所示的对话框，选择 Directories 选项卡。在 Show directories for 中选取 Include files，输入 C:\MATLAB6P5\EXTERN\INCLUDE，选取 Library files，输入 C:\MATLAB6P5\EXTERN\LIB\WIN32\MICROSOFT\MSVisual C++60。

（2）在“Project->Setting”菜单中选取 Link 选项卡，在 Object/library modules 中输入 libmat.lib 和 libmx.lib（中间以空格隔开），如图 6-4 所示。

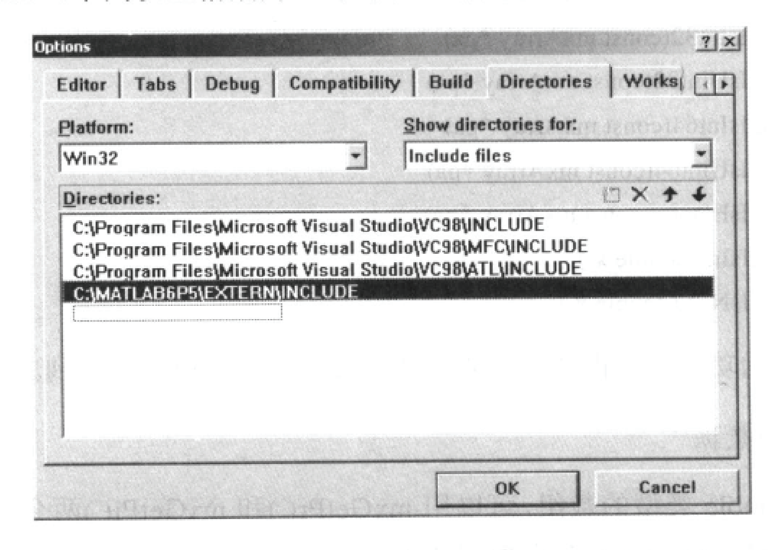


图 6-3　MATLAB 的 MAT API 环境设置添加头文件路径

（3）通过 MAT 文件实现 MATLAB 和 Visual C++数据共享的思路是将计算量大的操作交由 MATLAB 处理，将结果保存在 MAT 文件中，然后在 Visual C++工程中读取 MAT 数据。这样做的意义是：把计算过程留给 MATLAB，而 C++对计算结果进行分析或者可视化处理，即通过 MAT 数据文件共享以利用 MATLAB 强大的数据计算和处理能力。

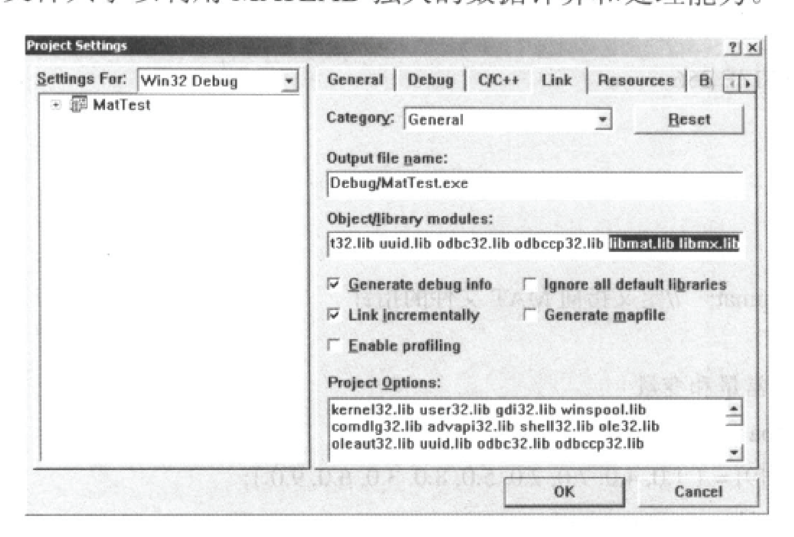


图 6-4　MATLAB 的 MAT API 环境添加库文件

6.4　实　　例

下面通过两个具体的实例来阐述 MAT 文件的使用，一个说明 MAT 文件如何创建，以及在 MATLAB 系统中如何查看，另一个说明在 Visual C++中如何读取已有的 MAT 文件并显示。

【例 6-1】在 Visual C++环境中通过 MAT API 创建一个能够被导入到 MATLAB 系统的 MAT 文件。以 C:\MATLAB6p5\extern\ examples\eng_mat 自带的 matcreate.c 为基础建立工程，其目的是验证 MAT API 中几个函数，如 matClose( )、matOpen( )、matGetVariable( )、matPutVariable( )等的使用。具体包含以下步骤：

（1）在 Visual C++中创建新工程，选择 Win32 Console Application，工程名为“MatCreateTest”。在 Step 1 of 1 选择 An empty project，其余选项均采用默认设置。

（2）将位于 C:\MATLAB6p5\extern\ examples\eng_mat 目录下的 matcreat.c 复制到当前工程目录中，并通过 Project 菜单中的 Add to project，添加 matcreat.c 到当前工程中。

（3）按 1.3 节 Visual C++调用 MAT 时环境设置的两个步骤，添加头文件路径和库文件到当前工程中。

（4）编译和运行工程。

以下列出了 matcreate.c 的全部代码，中文注释部分为手工添加。

```
/*
 * MAT-file creation program


 * Copyright 1986-2000 The MathWorks, Inc.
 */
/* $Revision: 1.13 $ */
#include <stdio.h>
```

```
#include <string.h>     /* For strcmp( ) */
#include <stdlib.h>     /* For EXIT_FAILURE, EXIT_SUCCESS */

#include "mat.h"      //MAT 文件 API 的头文件，必须添加

#define BUFSIZE 256

int main( )
{

   MATFile *pmat;   //定义指向 MAT 文件的指针

   //定义一些常量和变量
   mxArray *pa1, *pa2, *pa3;
   double data[9] = { 1.0, 4.0, 7.0, 2.0, 5.0, 8.0, 3.0, 6.0, 9.0 };
   const char *file = "mattest.mat";
   char str[BUFSIZE];
   int status;

   //打开一个 MAT 文件，如果不存在则创建一个新 MAT 文件，并检查返回值
   //如果打开失败，则返回
   printf("Creating file %s...\n\n", file);
   pmat = matOpen(file, "w");
   if (pmat == NULL)
   {
     printf("Error creating file %s\n", file);
     printf("(Do you have write permission in this directory?)\n");
     return(EXIT_FAILURE);
   }

   //创建一个名为 pa1 的矩阵，如果创建失败则返回
   pa1 = mxCreateDoubleMatrix(3,3,mxREAL);
   if (pa1 == NULL)
   {
        printf("%s : Out of memory on line %d\n", __FILE__, __LINE__);
        printf("Unable to create mxArray.\n");
        return(EXIT_FAILURE);
   }

   //创建一个名为 pa2 的矩阵，如果创建失败则返回
   pa2 = mxCreateDoubleMatrix(3,3,mxREAL);
   if (pa2 == NULL)
```

```
{
    printf("%s : Out of memory on line %d\n", __FILE__, __LINE__);
    printf("Unable to create mxArray.\n");
    return(EXIT_FAILURE);
}
//初始化矩阵 pa2
memcpy((void *)(mxGetPr(pa2)), (void *)data, sizeof(data));

//创建一个字符串类型的矩阵，如果创建失败则返回
pa3 = mxCreateString("MATLAB: the language of technical computing");
if (pa3 == NULL)
{
    printf("%s :  Out of memory on line %d\n", __FILE__, __LINE__);
    printf("Unable to create string mxArray.\n");
    return(EXIT_FAILURE);
}

//向 MAT 文件中写入变量 pa1，并命名为 LocalDouble
status = matPutVariable(pmat, "LocalDouble", pa1);
if (status != 0)
{
    printf("%s :  Error using matPutVariable on line %d\n", __FILE__, __LINE__);
    return(EXIT_FAILURE);
}

//向 MAT 文件中写入 pa2,并命名为 GlobalDouble，它是一个全局变量
status = matPutVariableAsGlobal(pmat, "GlobalDouble", pa2);
if (status != 0)
{
    printf("Error using matPutVariableAsGlobal\n");
    return(EXIT_FAILURE);
}

//向 MAT 文件中写入变量 pa3,并命名为 LocalString
status = matPutVariable(pmat, "LocalString", pa3);
if (status != 0)
{
    printf("%s :  Error using matPutVariable on line %d\n", __FILE__, __LINE__);
    return(EXIT_FAILURE);
}

/*
```

```
 * Ooops! we need to copy data before writing the array.   (Well,
 * ok, this was really intentional.) This demonstrates that
 * matPutVariable will overwrite an existing array in a MAT-file.
 */

//初始化变量 pa1
memcpy((void *)(mxGetPr(pa1)), (void *)data, sizeof(data));
//向 MAT 文件中写入变量 pa1，并命名为 LocalDouble
status = matPutVariable(pmat, "LocalDouble", pa1);
if (status != 0)
{
    printf("%s :  Error using matPutVariable on line %d\n", __FILE__, __LINE__);
    return(EXIT_FAILURE);
}

/*清除矩阵 */
mxDestroyArray(pa1);
mxDestroyArray(pa2);
mxDestroyArray(pa3);

//关闭 MAT 文件，如果出错则返回
if (matClose(pmat) != 0)
{
  printf("Error closing file %s\n",file);
  return(EXIT_FAILURE);
}

/*
 * 重新打开 MAT 文件，并验证文件的内容。如果打开失败则返回
 */
pmat = matOpen(file, "r");
if (pmat == NULL)
{
  printf("Error reopening file %s\n", file);
  return(EXIT_FAILURE);
}

/*
 * 读取变量 LocalDouble
 */
pa1 = matGetVariable(pmat, "LocalDouble");
if (pa1 == NULL)
```

```
{
  printf("Error reading existing matrix LocalDouble\n");
  return(EXIT_FAILURE);
}
//验证变量的总维数
if (mxGetNumberOfDimensions(pa1) != 2)
{
  printf("Error saving matrix: result does not have two dimensions\n");
  return(EXIT_FAILURE);
}

//读取变量 GlobalDouble，如果失败则返回
pa2 = matGetVariable(pmat, "GlobalDouble");
if (pa2 == NULL)
{
  printf("Error reading existing matrix GlobalDouble\n");
  return(EXIT_FAILURE);
}

//验证 pa2 是否在 MATLAB 的全局工作空间得到
if (!(mxIsFromGlobalWS(pa2))) {
  printf("Error saving global matrix: result is not global\n");
  return(EXIT_FAILURE);
}

//读取变量 LocalString
pa3 = matGetVariable(pmat, "LocalString");
if (pa3 == NULL)
{
  printf("Error reading existing matrix LocalString\n");
  return(EXIT_FAILURE);
}

//生成字符串
status = mxGetString(pa3, str, sizeof(str));
if(status != 0)
{
    printf("Not enough space. String is truncated.");
    return(EXIT_FAILURE);
}
```

```
    //验证字符的内容是否相符
    if (strcmp(str, "MATLAB: the language of technical computing"))
    {
      printf("Error saving string: result has incorrect contents\n");
      return(EXIT_FAILURE);
    }

    /* 退出前必须释放变量的内存 */
    mxDestroyArray(pa1);
    mxDestroyArray(pa2);
    mxDestroyArray(pa3);

    //关闭 MAT 文件
    if (matClose(pmat) != 0)
    {
      printf("Error closing file %s\n",file);
      return(EXIT_FAILURE);
    }

    printf("Done\n");
    return(EXIT_SUCCESS);

  }
```

编译后运行，运行界面如图 6-5 所示。在当前工程目录下可以发现，当前工程中生成了一个 MAT 文件 mattest.mat。

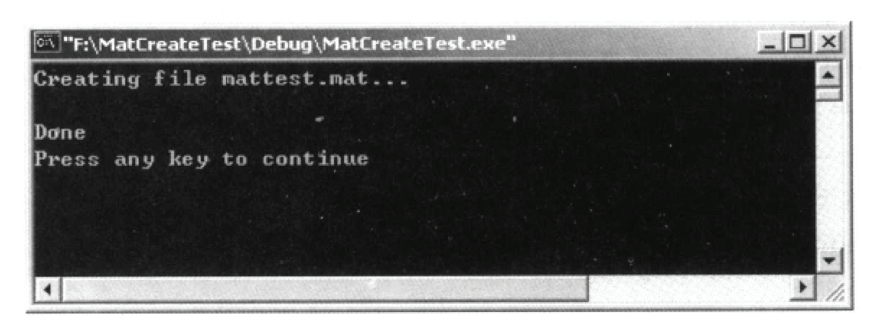


图 6-5　例 6-1 运行界面

下面验证 mattest.mat 在 MATLAB 系统中是否可以打开，以及其中变量的值正确与否。启动 MATLAB，单击 Workspace 中的 Load data file，找到位于当前工程目录下的 mattest.mat 文件。在 Workspace 中可以看到三个变量，也可以在 MATLAB 提示符下输入 whos，以查看 MATLAB 工作区间的变量。

```
>> whos
  Name                Size                    Bytes  Class
```

```
GlobalDouble        3x3                  72  double array
LocalDouble         3x3                  72  double array
LocalString         1x43                 86  char array

Grand total is 61 elements using 230 bytes
```

可见在 MAT 文件中生成了三个变量 GlobalDouble、LocalDouble 和 LocalString。双击工作区间中的 GlobalDouble 变量，弹出如图 6-6 所示的变量编辑窗口。

图 6-6　MATLAB 环境的变量编辑窗口

【例 6-2】 在 Visual C++环境中通过 MAT API 读取例 6-1 生成的 MAT 文件。以 C:\MATLAB6p5\extern\ examples\eng_mat 自带的 matdgns.c 为基础建立工程，其目的是验证 MAT API 中 matClose( )、matOpen( )、matGetNextVariable( )、matGetNextVariableInfo( )等函数的使用。具体包含以下步骤：

（1）在 Visual C++中创建新工程，选择 Win32 Console Application，工程名为 MatReadTest。在 Step 1 of 1 中选择 An empty project，其余选项均采用默认设置。

（2）将位于 C:\MATLAB6p5\extern\ examples\eng_mat 目录下的 matdgns.c 复制到当前工程目录中，并通过 Project 菜单中的 Add to project，选择 File 添加 matdgns.c 到当前工程中。

（3）按 1.3 节 Visual C++调用 MAT 时环境设置的两个步骤，添加头文件路径和库文件到当前工程中。

（4）编译和运行工程。

为方便程序运行，对 matdgns.c 中的 main( )函数进行了小的修改。以下列出了修改后的 matdgns.c 的全部代码，中文注释部分为手工添加。

```
//matdgns.c
#include <stdio.h>
#include <stdlib.h>
#include "mat.h"      //MAT   API 的头文件


int diagnose(const char *file)
{
  MATFile *pmat;
  char    **dir;
  const char *name;
```

```
int     ndir;
int     i;
mxArray *pa;

printf("Reading file %s...\n\n", file);

//打开 MAT 文件
pmat = matOpen(file, "r");
if (pmat == NULL)
{
  printf("Error opening file %s\n", file);
  return(1);
}

//得到当前 MAT 文件的变量目录并显示
dir = matGetDir(pmat, &ndir);
if (dir == NULL)
     {
      printf("Error reading directory of file %s\n", file);
     return(1);
     }
else
     {
     printf("Directory of %s:\n", file);
     for (i=0; i < ndir; i++)
         printf("%s\n",dir[i]);
     }

mxFree(dir);

 if (matClose(pmat) != 0)
{
  printf("Error closing file %s\n",file);
  return(1);
}

//重新打开 MAT 文件
pmat = matOpen(file, "r");
if (pmat == NULL)
{
  printf("Error reopening file %s\n", file);
  return(1);
```

```
}

//获取每个变量的变量头信息
printf("\nExamining the header for each variable:\n");
for (i=0; i < ndir; i++)
{
  pa = matGetNextVariableInfo(pmat, &name);
  if (pa == NULL)
    {
    printf("Error reading in file %s\n", file);
    return(1);
    }

//分析变量头，获得矩阵的维数
  printf("According to its header, array %s has %d dimensions\n",
   name, mxGetNumberOfDimensions(pa));
  //分析变量是否存在于 MATLAB 的全局工作区间
if (mxIsFromGlobalWS(pa))
     printf("   and was a global variable when saved\n");
  else
     printf("   and was a local variable when saved\n");
  mxDestroyArray(pa);
}

// 打开后重新关闭 MAT 文件
if (matClose(pmat) != 0)
{
  printf("Error closing file %s\n",file);
  return(1);
}

pmat = matOpen(file, "r");
if (pmat == NULL)
{
  printf("Error reopening file %s\n", file);
  return(1);
}

//通过 matGetNextVariable 获得矩阵变量的值
printf("\nReading in the actual array contents:\n");
for (i=0; i<ndir; i++)
{
```

```
        pa = matGetNextVariable(pmat, &name);
        if (pa == NULL) {
      printf("Error reading in file %s\n", file);
      return(1);
    }

    //分析矩阵数据，判断矩阵的维数以及变量类型
    printf("According to its contents, array %s has %d dimensions\n",
           name, mxGetNumberOfDimensions(pa));

    if (mxIsFromGlobalWS(pa))
   printf("    and was a global variable when saved\n");
    else
   printf("    and was a local variable when saved\n");

    mxDestroyArray(pa);
    }

    if (matClose(pmat) != 0)
    {
          printf("Error closing file %s\n",file);
          return(1);
    }

    printf("Done\n");
    return(0);
}

int main(int argc, char **argv)
{

    int result;

    //假设要打开的 MAT 文件名为 mattest.mat，并已在当前工作区目录中
    const char*file="mattest.mat";
    result=diagnose(file);

    return (result==0)?EXIT_SUCCESS:EXIT_FAILURE;

 }
```

编译后运行，运行结果如图 6-7 所示，发现 mattest.mat 中包含 GlobalDouble、 LocalString

和 LocalDouble 三个变量。其中，GlobalDouble 为 MATLAB 工作区中的全局变量，LocalString 和 LocalDouble 为局部变量。分析结果与例 6-1 完全一致。

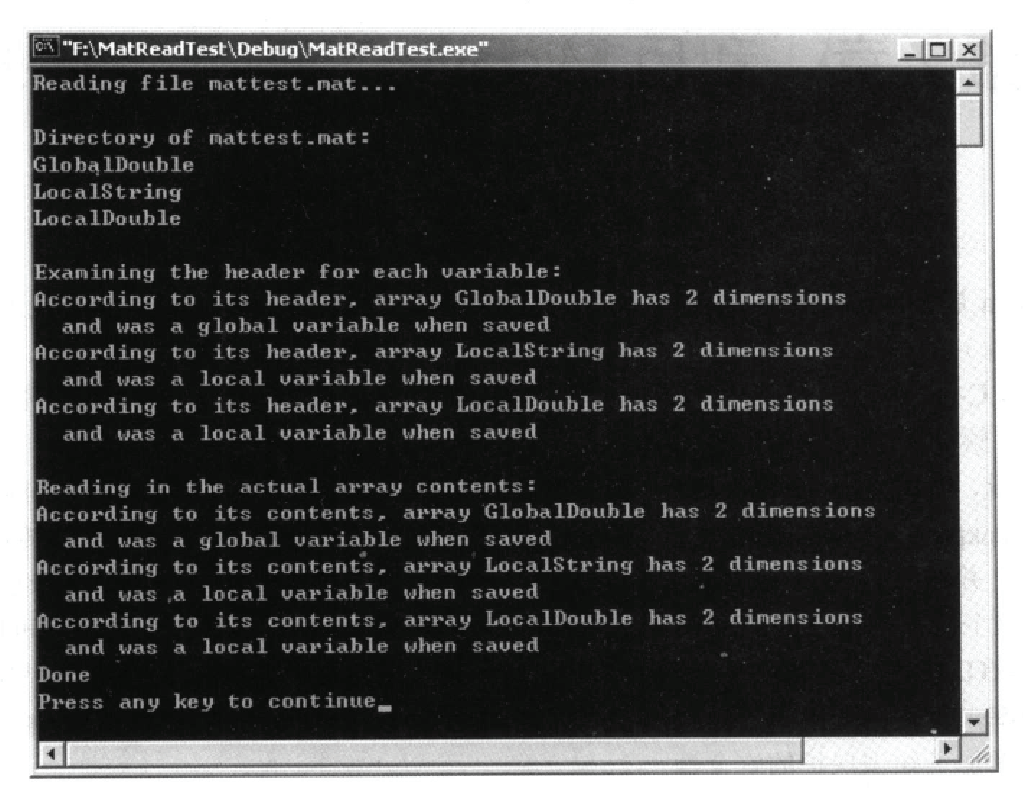


图 6-7　运行结果

6.5 小　　结

本节详细介绍了通过 MATLAB 自带的 MAT API 在 Visual C++创建工程和读取 MAT 文件的方法。由于 MAT API 完全屏蔽了 MAT 文件格式的细节，在 Visual C++中读取和创建 MAT 文件变得相当简单。通过 MAT 文件共享，可以利用 MATLAB 强大的数值计算和图像显示处理能力。因此，MAT 数据文件的共享提供了一种简单有效的混合编程方法。

第 7 章　利用 Mideva 实现混合编程

7.1　Mideva 简介

Mideva 是 MathTools 推出的一种 MATLAB 编译开发软件平台，提供对 MATLAB 程序文件（M 文件）的解释执行和开发环境支持，它集编辑、调试、编译和优化于一体。该软件有为 Borland C++、Visual C++和 Visual Basic 等编程语言开发的不同版本，目前其版本已经到了 4.5 版，软件大小仅为 8.5MB。

在编辑、调试方面 Mideva 比 MATLAB 强大，如语法突出显示，批量注释，可以用 step，watch，breakpoint 等各种调试手段。Mideva 最主要特点是能够将 M 语言的文件转化为 C 语言的代码，并通过 Visual C++和 Borland C++将其编译成可执行程序（.exe）或动态链接库（.dll）。这个特点也是 MathTools 之所以要推出该软件的出发点。因此，通过它可以很容易地实现 MATLAB 和 Visual C++的混合编程，而且具有混合编程的所有优点，既提高了 M 代码的复用率，又提高了代码的执行速度，并将 MATLAB 的 M 文件转换为二进制格式的可执行程序，增加了知识保护的安全性。

Matcom 是 Mideva 的内核，它是一个基于 C++矩阵函数库 Matrix<LIB>的一个 MATLAB 的 M 文件与 C、CPP 文件的转换程序。可以理解 Mideva 为 Matcom 的一个集成调试编译环境（界面）。在很多 MATLAB 图书中，对二者并不加以区分，本书中同样不加以区分。

Matcom 独立于 MATLAB 之外，但需要外部的 C++Compiler，在安装时需要指定 C++ Compiler 的位置，否则无法工作。安装时需要指定 MATLAB 的位置，主要是为了编译文件时找到需要的一些系统函数的路径。

因为现在 MathTools 公司已经被生产 MATLAB 的 MathWorks 公司兼并，目前 Mideva 已经被集成到 MATLAB 6.0 以后版本中，因此 MathWorks 不再提供单独的 Mideva 产品，但互联网上仍有很多以前的免费下载版本。由于 M 文件经过 Mideva 转换后生成的 C、CPP 代码具有简单、高效和易于理解的特点，可以在 Visual C++工程中灵活使用，因此本章仍进行较详细的介绍。

7.2　Mideva 的安装

Mideva 的安装与任何 Windows 下的应用程序安装一样，会出现一些标准 Windows 风格的安装对话框，但是安装过程中弹出的对话框较多，通常会出现一些小的问题。下面对一些常见的问题进行简要介绍。

在按照对话框的要求输入安装路径，系统拷贝完程序后，会弹出如图 7-1 所示的对话框，要求输入 License。这时可以将从 MathTools 公司申请到的 License 输入（注：现在很多网络上有 Mideva 的破解密码）。然后系统会弹出如图 7-2 所示的对话框，提示 Mideva 已经自动找到了 C/C++的编译器（以 Visual C++为例），选择“是（Y）”接受。

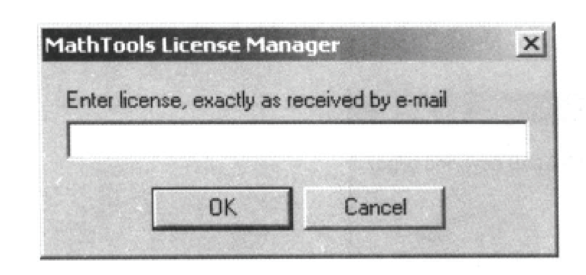


图 7-1　Mideva 的 License 对话框



图 7-2　Mideva 自动搜索 C/C++编译器对话框

然后，系统会弹出如图 7-3 所示的对话框，提示是否安装了 MATLAB。如果机器上已经安装了 MATLAB 则选择“是（Y）”，系统弹出如图 7-4 所示的对话框，提示是否自动搜索 MATLAB，同样选择“是（Y）”。

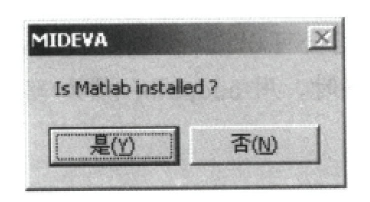




图 7-3　MATLAB 是否安装提示对话框



图 7-4　是否自动搜索 MATLAB 对话框

这时，通常会出现如图 7-5 所示的出错告警对话框，提示安装 Matcom 在搜索 MATLAB 路径时找不到 matcom.m。出错的原因是 MATLAB 6.x 的 Windows 版本中 MATLAB.exe 不在 MATLAB\bin 下，而在 MATLAB\bin\win32 下，所以在安装 Matcom 时，把 bin\win32\MATLAB.exe 移动到 bin\，即上移一层目录，安装 Matcom 后再移动回去即可。

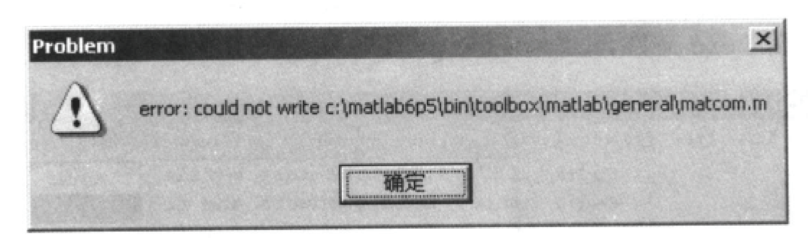


图 7-5　安装过程中常见的出错告警对话框

从 Windows“开始”栏“程序”组启动 Mideva，会出现如图 7-6 所示的对话框，提示为创建必要的路径，需要在 MATLAB 环境中执行提示框中的 MATLAB 命令。单击“确定”按钮，对话框中的 MATLAB 命令会出现在 Mideva 的窗口中，复制这些 MATLAB 命令，然后启动 MATLAB，在 MATLAB 环境中执行这些命令后退出 MATLAB。此时，关闭 Mideva 后重新启动 Mideva 将不再出现任何提示或告警信息，它表明 Mideva 已经成功安装。

为了验证 Mideva 是否安装，可以在 Mideva 中执行 magic( )函数，若能正确运行则表明安装成功。

```
» magic(5)
ans (5x5)=25 double elements real (200 bytes) =
17  24   1   8  15
23   5   7  14  16
6   13  20  22
10  12  19  21   3
```

11 18 25 2 9

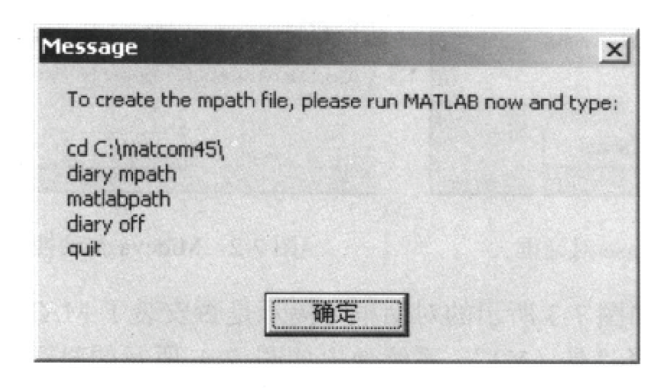


图 7-6 提示需要在 MATLAB 环境中执行的命令

注：是否安装有 MATLAB 对 Matcom 没有什么影响，完全可以选择没有安装 MATLAB，仍然可以编译大多数文件。需要 MATLAB\toolbox 下的文件时，用 addpath( )添加路径或者拷贝到当前目录下即可。

7.3 Mideva 环境下 M 文件到 dll/exe 文件的转换

在 MATLAB 环境中可以由 M 文件生成执行程序，但此 exe 文件仍不能脱离 MATLAB 的后台支持。利用 Matcom 则可以生成不再需要 Matcom 环境支持的 exe 文件。其生成方法相当简便，直接利用 Matcom 菜单 File /Compile to exe/（如图 7-7 所示），然后选择要生成的 M 文件，Matcom 便会自动完成 exe 文件的编译与链接。编译完后就可以在 DEBUG 目录中找到与 M 文件同名的 exe 文件。

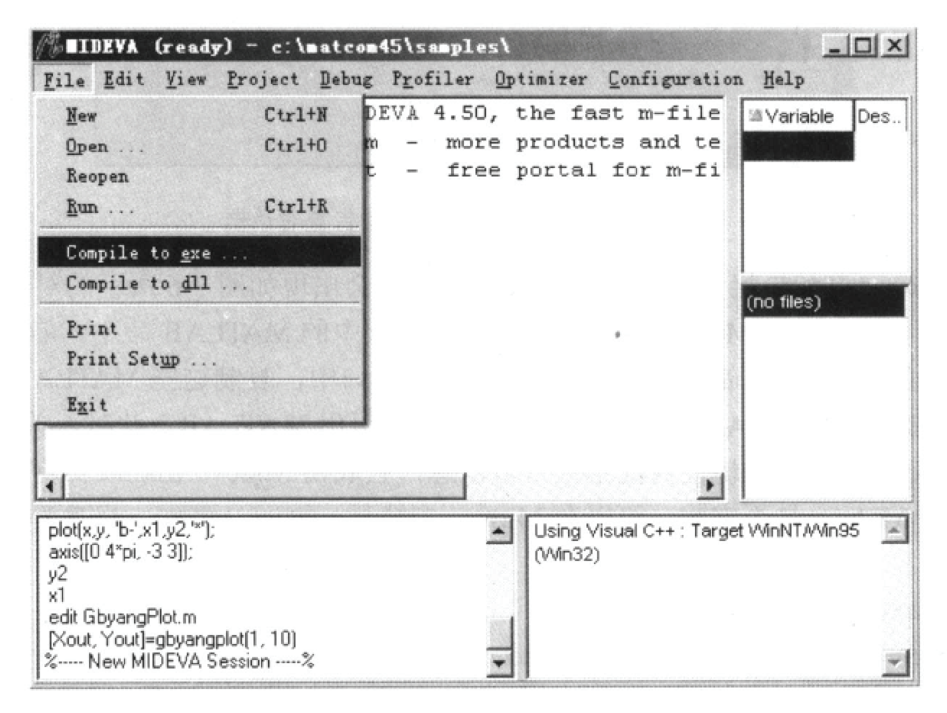


图 7-7 Mideva 运行时部分界面

这一功能只针对 Visual C++开发，dll 的生成方法如同生成 exe 文件的方法，它可以生成

针对 VB，Excel，Delphi 等的动态链接库文件，使用起来非常方便。生成 dll 的过程中同样要生成 cpp 文件。MathTools 公司建议用户：如果是用 Visual C++开发程序，那么最好直接使用 Matcom 编译生成的 cpp 文件，而不是直接使用 dll 文件。

另外值得说明的一点是：Mideva 环境下绘制的图形，在 Control 菜单下包含 OpenGL 和 Zoom 等子菜单，因此，可以通过菜单进行调整。如图 7-8 所示是一条简单的正弦曲线，与 MATLAB 环境下绘制的图形外观差异不大。可以通过菜单进行调整，调整后的图形如图 7-9 所示。

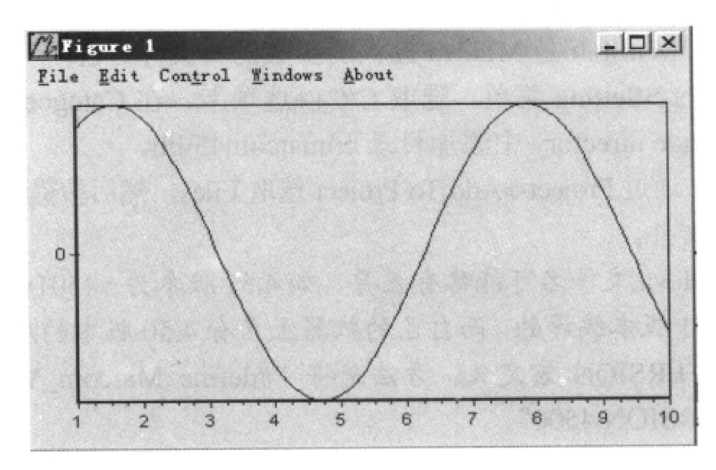


图 7-8　Mideva 环境下的图形绘制

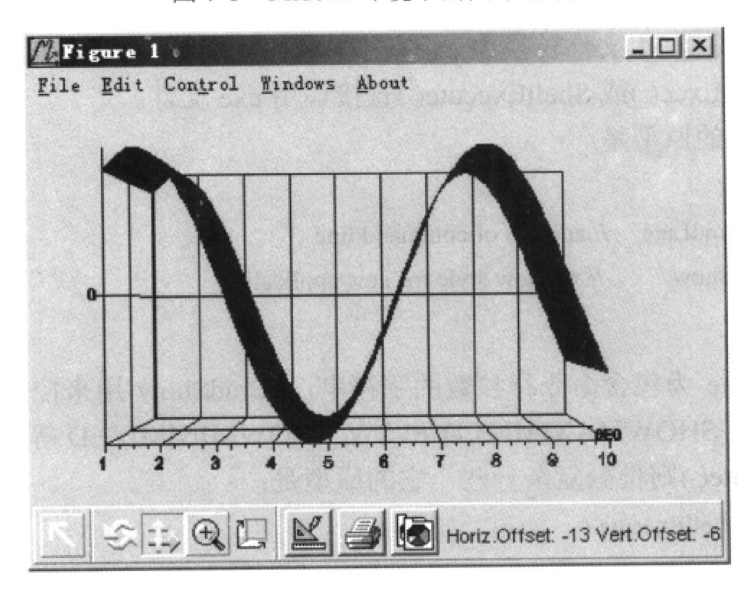


图 7-9　Mideva 下绘制的图形通过菜单进行调整

7.4　Visual C++环境下使用 Mideva 混合编程

Mideva 提供的功能相当强大，因为它包含了近千个 MATLAB 的基本功能函数，通过必要的设置就可以直接实现与 C++的混合编程，而不必再依赖 MATLAB。同时，Mideva 还提供编译转换功能，能够将 MATLAB 函数或编写的 MATLAB 程序转换为 C++形式的 dll，从而实现脱离 MATLAB 环境对 MATLAB 函数和过程的有效调用，这样就有可能实现对 MATLAB 工具箱函数的利用。

7.4.1 混合编程环境的设置

Mideva 安装时会提示选取 C/C++编辑器，还需要指定 MATLAB，主要是为了编译文件中需要的一些系统函数来找到路径用的。Mideva 提供了易于使用的接口，可直接通过菜单 File->Compile to exe 或 Compile to dll 将 M 文件转换为可执行文件，同时自动生成了相应的 C/CPP 文件。

为了借助 Visual C++强大的集成调试环境，以便将生成的 C/CPP 代码添加到 Visual C++的工程中，必须先进行必要的设置。主要有以下几步：

（1）添加头文件。Matlib.h 是 Mideva 提供的函数的头文件，它位于 c:\matcom45\lib 目录下。方法是通过 Project->Setting 菜单，选取 C/C++选项卡，在 Category 中选取 Preprocessor，然后在 Additional include directory 中添加目录 c:\matcom45\lib。

（2）添加库文件。通过 Project->Add To Project 选取 Files，然后按提示选取 c:\matcom45\lib 目录下的 v4500v.lib 文件。

注意：不同的 Mideva 文件名可能略有差异，如 4.51 版本为 v4501v.lib。如果读者从网上发现有的程序是用 4.51 版本编译的，而自己的机器上只有 4.50 版本的库文件，这时可以修改 matlib.h 中 Matcom_VERSION 宏定义。方法是将"#define Matcom_VERSION 4501"改为"#define Matcom_VERSION 4500"。

7.4.2 通过外壳函数调用

另外，Mideva 可将 M 文件转换为 exe 文件，也实现了一种混合编程的途径。实现方法是利用外壳函数 WinExec( )或 ShellExecute( )直接调用 exe 文件。

函数 WinExec( )的原型是：

```
UINT WinExec(
  LPCSTR lpCmdLine,   // address of command line
  UINT nCmdShow       // window style for new application
);
```

其中，lpCmdLine 为包含命令行参数的字符串，nCmdShow 用来控制窗口状态，常见的如 SW_SHOW，SW_SHOWMAXIMIZED 和 SW_SHOWMINIMIZED 等。

函数 ShellExecute( )则稍微复杂一些，它的原型是：

```
HINSTANCE ShellExecute (
        HWND hwnd,
        LPCTSTR lpOperation,
        LPCTSTR lpFile,
        LPCTSTR lpParameters,
        LPCTSTR lpDirectory,
        INT nShowCmd
        );
```

其中，hwnd 是指向父窗口的窗口句柄；lpOperation 为一字符串，指定要执行的操作，这里应设为 NULL。lpFile 为文件名，即通过外壳函数调用的 exe 文件的名字。lpParameters 为可执行文件的参数，而 lpDirectory 为 exe 文件所在的目录；nShowCmd 是窗口的显示状态，与

WinExec 的 nCmdShow 含义相同。

通过外壳函数直接调用 exe 的方法，实现简单，且无须链接头文件和库，但一般在程序中不能接收返回值，灵活性差。

7.5 Matrix<LIB>

通过 Mideva 自动地将 M 文件转换为 C、CPP 文件，然后将生成的 C、CPP 代码拷贝到 Visual C++工程中，从而实现 Mideva 和 Visual C++的混合编程，可以降低应用程序的开发难度。Matcom 是 Mideva 内核的 C++编译器，它可以生成 C-MEX 和独立应用程序。实际上，Matcom 包含了一组称为 Matrix<LIB>的 C++库。它是 MathTools 公司开发的一个矩阵数学库，提供了一个双精度 Matrix 类型——Mm，它可以是复数矩阵、实数矩阵、稀疏矩阵甚至 n 维矩阵。这个库共有 600 多个函数和重载的操作符，涉及线性代数、多项式数学、信号处理、文件输入/输出、图像处理、绘图等方面。这些函数都经过仔细优化，以提供更好的矩阵操作性能。

通过 Mideva 实现混合编程的实质是利用了 Matrix<LIB>中的库函数。Matrix<LIB>的函数基于矩阵类型，大多数函数原型类似于 MATLAB 函数，而且添加必要的头文件和库文件后可以在 Visual C++环境下编译运行。至于 Matrix<LIB>的详细内容以及按 Matrix<LIB>语法实现混合编程，请参阅第 8 章。

7.6 混合编程实例

下面通过两个具体的实例说明通过 Mideva 实现混合编程的方法（假设机器上已经正确安装了 Mideva）。

【例 7-1】启动 Mideva，编辑一个名为 gbyang 的 M 文件，其功能是在一个窗口中绘制 4 个图形。M 文件如下：

```
t=0: pi/20: 2*pi;
[x, y]=meshgrid(t);

subplot(2,2,1);
plot(sin(t),cos(t));
axis equal;

subplot(2,2,2);
z=sin(x)+cos(y);
plot(t,z);
axis([0 2*pi -2 2]);

subplot(2,2,3);
z=sin(x);
z=sin(x).*cos(y);
plot(t,z);
```

```
axis([0 2*pi -1 1]);

subplot(2,2,4);
z=(sin(x).^2)-(cos(y).^2);
plot(t,z);
axis([0 2*pi -1 1]);
```

通过 Mideva 菜单，将 gbyang.m 文件转换为 exe 文件，它会同时生成 cpp 文件。打开 gbyang.cpp，代码如下：

```
dMm(t); dMm(x); dMm(y); dMm(z);

call_stack_begin;
t = colon(0.0,pi/20.0,2.0*pi);
/*[x,y] = */meshgrid(t,i_o,x,y);
subplot(2.0,2.0,1.0);
plot((CL(sin(t)),cos(t)));
axis(TM("equal"));
subplot(2.0,2.0,2.0);
z = sin(x)+cos(y);
plot((CL(t),z));
axis((BR(0.0),2.0*pi,-2.0,2.0));
subplot(2.0,2.0,3.0);
z = sin(x);
z = times(sin(x),cos(y));
plot((CL(t),z));
axis((BR(0.0),2.0*pi,-1.0,1.0));
subplot(2.0,2.0,4.0);
z = (power(sin(x),2.0))-(power(cos(y),2.0));
plot((CL(t),z));
axis((BR(0.0),2.0*pi,-1.0,1.0));

call_stack_end;
```

生成的 gbyang.h 文件代码如下：

```
#ifndef __gbyang_h
#define __gbyang_h
⋮
#endif // __gbyang_h
```

请仔细对照生成的 gbyang.cpp 代码与 gbyang.m 代码，可以发现它们特别类似，基本上句句对应。如 M 文件中的“[x, y]=meshgrid(t);”经转换后变为“/*[x,y] = */meshgrid(t,i_o,x,y);”其中“/*”和“*/”间的内容为自动生成的注释，i_o 为多个输出函数中作为输入、输出函数的分隔符。另外，生成的 cpp 中的“call_stack_begin；”和“call_stack_end；”可以注释掉

并不影响编译，但最好不要去掉，以保持代码的完整性。

接下来，我们建立一个 Visual C++新工程，选择 MFC AppWizard (exe)，工程名为 MidevaTest。在 step 1 选择建立 Single Document，其余各项均采用默认设置，点击“Finish”。然后进行以下几步操作：

按 7.4.1 节中（1）和（2）进行环境设置。

添加以下菜单项，菜单项的属性设置见表 7-1，并在视图类添加相应的事件响应函数。

表 7-1　菜单项的属性设置

Caption	ID	Prompt
&CPP 调用	ID_MIDEVA_CPP	调用 Mideva 生成的 CPP 代码
&EXE 调用	ID_MIDEVA_EXE	通过外壳函数调用 Mideva 生成的 EXE

在 MidevaTestView.cpp 开始部分添加#include "matlib.h"，以将头文件包含到工程中。

在 void CMidevaTestView::OnMidevaCpp( )函数中先添加 initM(Matcom_VERSION)，然后将 gbyang.m 函数经 Mideva 自动转换生成的 CPP 文件中所有语句拷贝过来，最后添加 exitM( )。注意：initM( )和 exitM( )是 Mideva 工程所必需的，前一句在初始化类库时使用，后一句为结束类库使用。

将 Mideva 转换生成的 gbyang.exe 复制到本工程的目录下。

在 void CMidevaTestView::OnMidevaExe( )函数中先添加以下语句：

```
WinExec("gbyang.exe", SW_SHOW);
```

以下是新添加菜单的响应事件代码

```
void CMidevaTestView::OnMidevaCpp( )
{
  // TODO: Add your command handler code here


  initM(Matcom_VERSION);
  dMm(t); dMm(x); dMm(y); dMm(z);


#line 1 "c:/matcom45/samples/gbyang.m"
  call_stack_begin;
#line 1 "c:/matcom45/samples/gbyang.m"
  _ t = colon(0.0,pi/20.0,2.0*pi);
#line 2 "c:/matcom45/samples/gbyang.m"
  _ /*[x,y] = */meshgrid(t,i_o,x,y);


#line 4 "c:/matcom45/samples/gbyang.m"
  _ subplot(2.0,2.0,1.0);
#line 5 "c:/matcom45/samples/gbyang.m"
  _ plot((CL(sin(t)),cos(t)));
#line 6 "c:/matcom45/samples/gbyang.m"
```

```
  _ axis(TM("equal"));

#line 8 "c:/matcom45/samples/gbyang.m"
  _ subplot(2.0,2.0,2.0);
#line 9 "c:/matcom45/samples/gbyang.m"
  _ z = sin(x)+cos(y);
#line 10 "c:/matcom45/samples/gbyang.m"
  _ plot((CL(t),z));
#line 11 "c:/matcom45/samples/gbyang.m"
  _ axis((BR(0.0),2.0*pi,-2.0,2.0));

#line 13 "c:/matcom45/samples/gbyang.m"
  _ subplot(2.0,2.0,3.0);
#line 14 "c:/matcom45/samples/gbyang.m"
  _ z = sin(x);
#line 15 "c:/matcom45/samples/gbyang.m"
  _ z = times(sin(x),cos(y));
#line 16 "c:/matcom45/samples/gbyang.m"
  _ plot((CL(t),z));
#line 17 "c:/matcom45/samples/gbyang.m"
  _ axis((BR(0.0),2.0*pi,-1.0,1.0));

#line 19 "c:/matcom45/samples/gbyang.m"
  _ subplot(2.0,2.0,4.0);
#line 20 "c:/matcom45/samples/gbyang.m"
  _ z = (power(sin(x),2.0))-(power(cos(y),2.0));
#line 21 "c:/matcom45/samples/gbyang.m"
  _ plot((CL(t),z));
#line 22 "c:/matcom45/samples/gbyang.m"
  _ axis((BR(0.0),2.0*pi,-1.0,1.0));

  call_stack_end;
  exitM( );
}

void CMidevaTestView::OnMidevaExe( )
{
  // TODO: Add your command handler code here
  WinExec("gbyang.exe", SW_SHOW);
//ShellExecute(NULL, NULL, "gbyang.exe", NULL, NULL, SW_SHOW);
}
```

编译后运行，运行界面如图 7-10 所示。分别单击“CPP 调用”和“EXE 调用”，结果相同，都如图 7-11 所示。唯一的差别在于：“EXE 调用”时会闪出一个 DOS 窗口，它与直接双击 gbyang.exe 文件的执行效果完全一样。

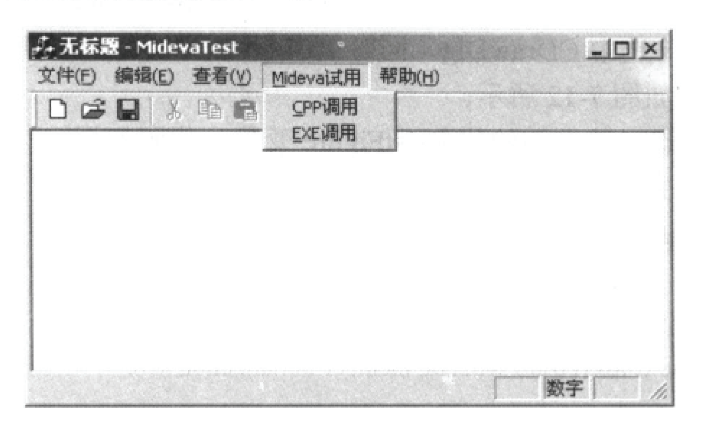


图 7-10　运行界面

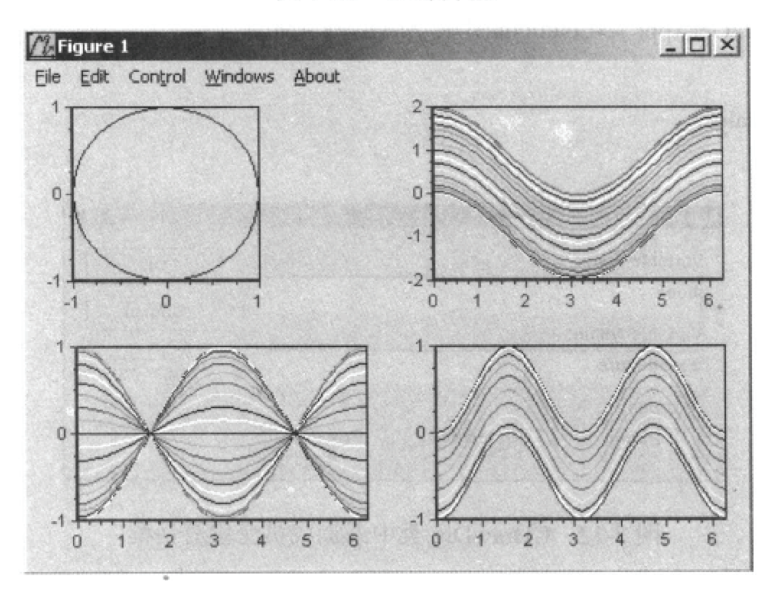


图 7-11　运算结果

【例 7-2】下面是一个基于对话框类型的 MFC 应用程序。它通过 Mideva 和 Visual C++ 的混合编程实现在对话框窗口中绘制 2D 和 3D 图形。具体包含以下步骤：

启动 Visual C++，建立一个新工程，选择 MFC AppWizard(exe)，工程名为 draw。在“Step 1 of 1”选择“Dialog based”,即生成一个基于对话框的 MFC 应用程序，其余各项采用默认设置。

使用库的头文件 matlib.h。首先按 7.4.1 节中“混合编程环境的设置”的步骤，添加必要的头文件和库函数的路径，并在 drawDlg.cpp 中添加头文件，如下所示：

```
// drawDlg.cpp : implementation file
//

#include "stdafx.h"
#include "draw.h"
#include "drawDlg.h"
```

```
#include "matlib.h"
```

在 CDrawDlg 类中添加 bool 类型的 public 成员变量 ExistFigure，方法是在 Workspace 中单击 ClassView 选项卡，选中 CDrawDlg，同时单击右键，选择 Add Member Variable，出现添加成员变量对话框，如图 7-12 所示。

在 CDrawDlg 的构造函数中初始化 ExistFigure 变量，代码如下：

```
CDrawDlg::CDrawDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CDrawDlg::IDD, pParent)
{
  //{{AFX_DATA_INIT(CDrawDlg)
        // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
  m_hIcon = AfxGetApp( )->LoadIcon(IDR_MAINFRAME);

  ExistFigure=false;
  }
```

图 7-12　CDrawDlg 类中添加成员变量对话框

在 IDD_DRAW_DIALOG 对话框资源中删除自动生成的“取消”按钮，添加两个 Button 控件和一个 Picture 控制，并按图 7-13 所示调整好控制的大小和布局，然后按表 7-2 设置控件的 Caption 和 ID 属性。

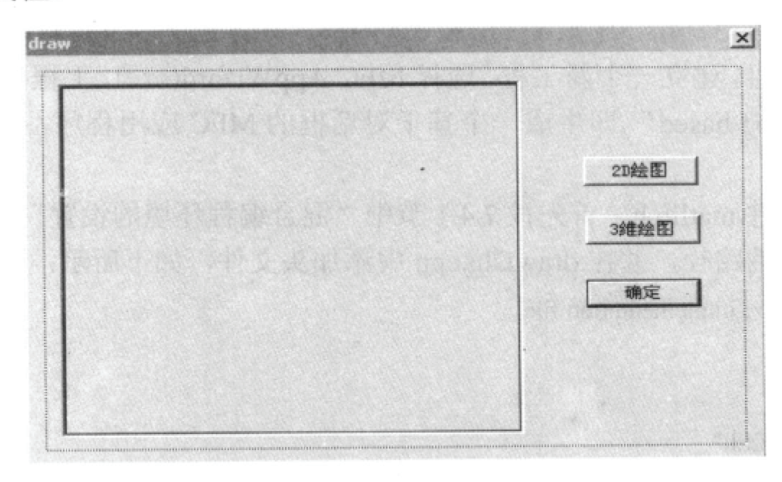


图 7-13　窗口布局对话框

表 7-2 新增控件的属性设置

Caption	ID	备　注
2D 绘图	IDC_2DDRAW	Button 控件
3D 绘图	IDC_3DDRAW	Button 控件
	IDC_PIC	Picture 控件

通过 ClassWizard 添加按钮控件的响应事件，代码如下：

```
void CDrawDlg::On2ddraw( )
{
  // TODO: Add your control notification handler code here

  initM(Matcom_VERSION);    //初始化类库

  //根据标志判断是否已有图形,如有先清除
  if (ExistFigure==true)
  {
        clf(1);
        ExistFigure=false;
  }

  //以下代码绘制二维图形

  Mm t;  //定义变量
     t = colon(0.0,0.01,2.0*pi);

  //设置绘图的位置等属性
 CWnd *p1=NULL;
  p1=(CWnd *)GetDlgItem(IDC_PIC);
  Mm plothandle=winaxes(p1->m_hWnd);
 Mm pos=(BR(20),20,200,100);
 set(plothandle,TM("RealPosition"),pos);

  //绘图
 polar((CL(t),abs(times(sin(2.0*t),cos(2.0*t)))));
  drawnow( );

  //设置是否绘图标志
  ExistFigure=true;

  exitM( ); //退出类库
 }
```

```
void CDrawDlg::On3ddraw( )
{
  // TODO: Add your control notification handler code here

  initM(Matcom_VERSION); //初始化类库

  //根据标志判断是否已有图形,如有先清除
  if (ExistFigure==true)
  {
        clf(1);
        ExistFigure=false;
  }

  //以下代码绘制三维图形

  dMm(y);
 dMm(z); dMm(x1); dMm(y1);

 y = colon(8.0,-0.5,-8.0);
   meshgrid(y,y,i_o,x1,y1);
   z = power(x1,2.0)+power(y1,2.0);

   //设置绘图的位置等属性
   CWnd *p1=NULL;
   p1=(CWnd *)GetDlgItem(IDC_PIC);
   Mm plothandle=winaxes(p1->m_hWnd);

  Mm pos=(BR(20),20,100,50);
   set(plothandle,TM("RealPosition"),pos);
   //set(plothandle,(CL(TM("Color")), TM("white")));      //去掉本行前的注释符会出现如图 7-15 所示
的结果

   //绘图
   mesh((CL(z)));
   drawnow( );

   //设置是否绘图标志
   ExistFigure=true;

   exitM( );//退出类库

 }
```

编译后运行，出现如图 7-14 所示的运行界面。分别单击“2D 绘图”和“3 维绘图”按钮，会出现如图 7-15 和图 7-16 所示的实验结果。在去掉代码中修改图形属性代码前的注释符后，重新编译链接，单击“3 维绘图”按钮则出现如图 7-17 所示的实验结果。

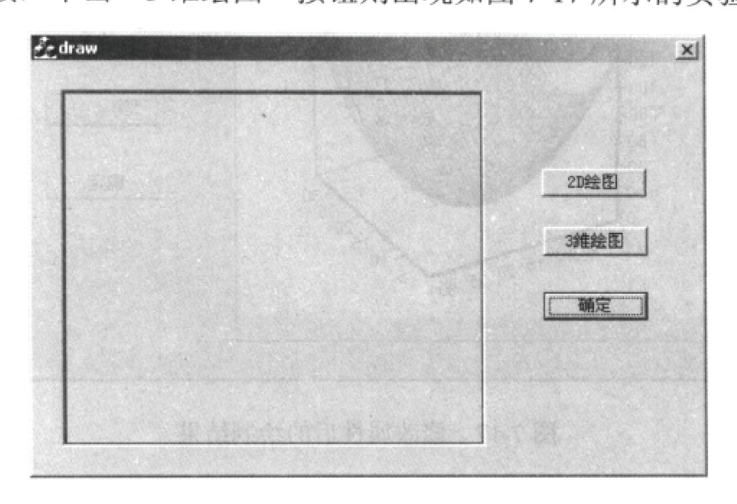


图 7-14　运行界面

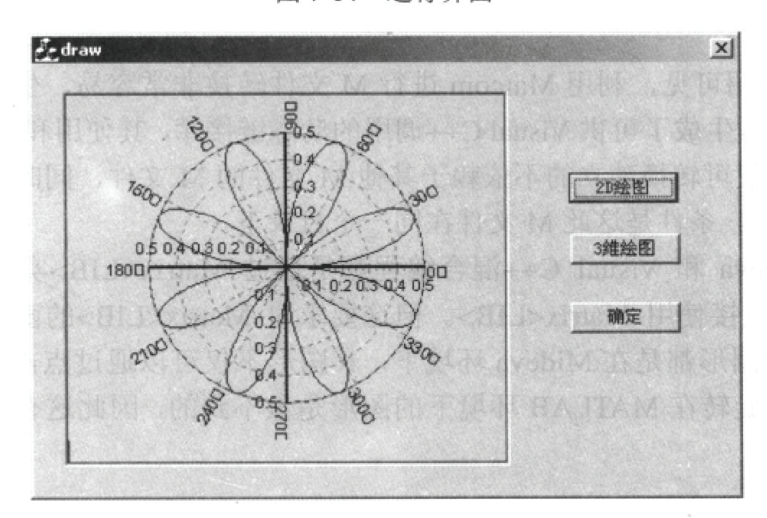


图 7-15　2D 绘图结果

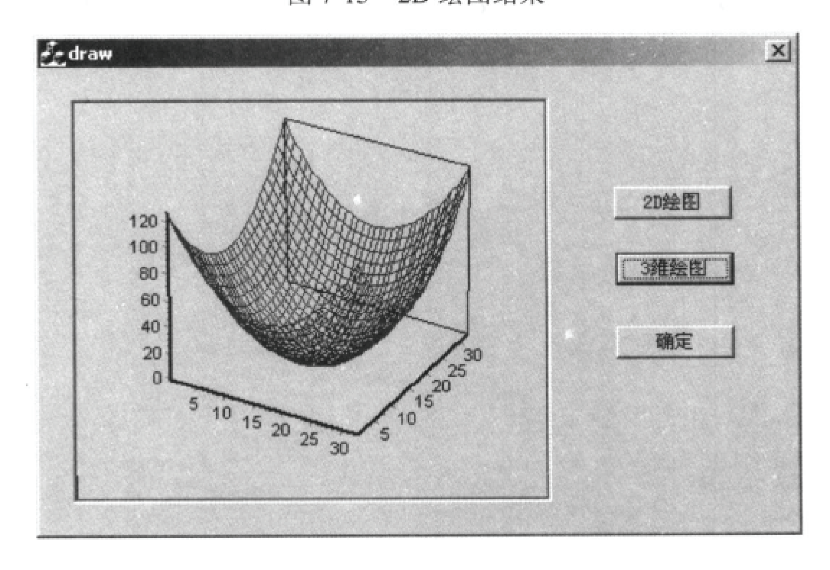


图 7-16　三维绘图结果

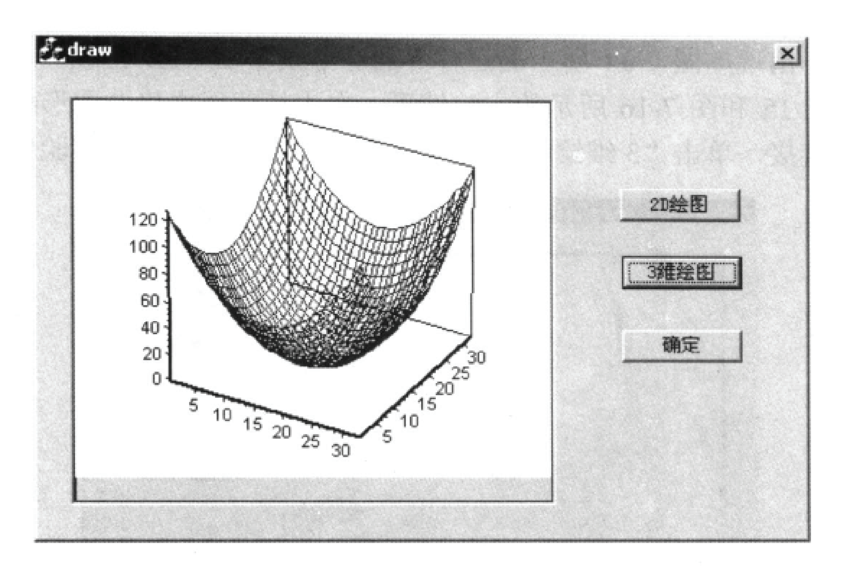


图 7-17　修改属性后的绘制结果

7.7　小　　结

通过本章的介绍可见，利用 Matcom 进行 M 文件转换非常容易，生成的代码可读性很好，以上的转换同时生成了可供 Visual C++调用的动态链接库，其使用和一般的动态库一样。需指明 Matcom 不仅可转换独立的不依赖于其他 M 文件的 M 文件，同时可转换调用其他 M 文件的 M 文件嵌套。条件是这此 M 文件在同一个目录下。

实际上，Mideva 和 Visual C++混合编程都是通过 Matrix<LIB>实现的，因此也可在 MFC 应用程序中直接使用 Matrix<LIB>。但这要求对 Matrix<LIB>的函数较为熟悉。值得一提的是，绘制的图形都是在 Mideva 环境下，其图形不仅可以通过点击鼠标任意放大，还可以任意旋转，而旋转在 MATLAB 环境下的图形是做不到的，因此这有助于从各个侧面观察曲线/曲面。

第 8 章　利用 Matrix<LIB>实现混合编程

8.1　Matrix<LIB>简介

第 7 章介绍了 Mideva 提供的一种实现混合编程的方法：通过 Mideva 自动将 M 文件转换为 C、CPP 文件，然后将生成的 C、CPP 代码拷贝到 Visual C++工程中，从而降低开发难度。Matcom 是 Mideva 内核的 C++编译器，它可以生成 C-MEX 和独立应用程序。实际上，Matcom 包含了一组称为 Matrix<LIB>的 C++库。它是 MathTools 公司开发的一个矩阵数学库，提供了一个双精度 Matrix 类型——Mm，它可以是复数矩阵、实数矩阵、稀疏矩阵甚至 n 维矩阵。这个库共有 600 多个函数和重载的操作符，涉及线性代数、多项式数学、信号处理、文件输入/输出、图像处理、绘图等方面。这些函数都经过仔细优化，以提供更好的矩阵操作性能。

8.2　Matrix<LIB>与 Visual C++混合编程

在 Visual C++工程中直接按照 Matrix<LIB>的语法编程，即可实现高效的编程。这是因为 Matrix<LIB>的函数基于矩阵类型，大多数函数原型类似于 MATLAB 函数。实际上，Matrix<LIB>是将一些很有用的 MATLAB 函数封装成 C++的库文件，库以一组 Windows 动态链接库的形式提供，因此生成的可执行文件很小。

8.2.1　Matrix<LIB>的安装

因为 Matrix<LIB>是 Matcom 的内核，而 Matcom 是 Mideva 的 C/C++编译器。因此，当计算机上安装了 Mideva 以后，即自动安装了 Matrix<LIB>矩阵数学库。

8.2.2　Visual C++环境配置

Matrix<LIB>是一个 C++的库。为了将库文件集成到 C++的工程文件中，必须对 Visual C++工程进行一些必要的环境设置。具体需要执行如下操作：

1. 将库文件加到 C++的工程中

Matrix<LIB>必须加入到 C++的工程文件中，以便 C++编译器能够将它链接到最后的执行文件中去。在 Visual C++中利用菜单命令 “Project/Add to Project / Files ” 将 v4500v.lib 加入到 C++工程中。

2. 使用库的头文件 matlib.h

它描述 Matrix<LIB>中用到的数据类型、函数原型以及常量。Visual C++工程中需要添加 #include "matlib.h"，并同时告诉编译器文件 matlib.h 所在的路径，即通过菜单项 “Project/

Settings/C, C++/Preprocessor / Additional include subdirectories”添加 c:\matcom45\lib 来完成此项工作。

8.2.3 初始化库

在调用 Matrix<LIB>之前要使用函数 initM(Matcom_VERSION) 来初始化类库调用，并用 exitM( ) 函数来结束类库调用。Matcom_VERSION 是一个在 matlib.h 中定义了的常量，它保证动态链接库与 matlib.h 相匹配。ExitM( )应该是程序中返回命令 return 之前的最后一条语句。

1. 生成矩阵

Matrix<LIB>定义矩阵类型的关键字为 Mm。如果需要定义矩阵 a, b 和 x，则可以通过添加下列语句构造矩阵 a, b 和 x:

```
Mm a;
Mm b;
Mm x;
```

新构造的矩阵 a, b, x 还是空矩阵。上面的三行语句应该添加到 initM( )和 exitM( )之间。

2. 设置矩阵元素

设置矩阵 a 为 3×3 的随机矩阵，然后给 b 赋值。

```
a=rand(3); b=zeros(3, 1);
b.r(1, 1)=3; b.r(2, 1)=-1; b.r(3, 1)=5;
```

值得一提的是，在给 a 赋值前并没有给 a 预定义维数和分配内存。重载后的运算符“=”可以处理这些操作。而矩阵 b 预先通过 b=zeros(3, 1)指定了维数，使用了直接访问的成员函数 r(row, col)。

所有的内存管理和矩阵维数的重新分配都是自动的，创立矩阵变量时一般很少需要分配矩阵维数。

3. 访问矩阵的元素

可以用 r(row, col)访问矩阵的单个元素，其中 row, col 分别为矩阵元素的行、列下标。另外两个函数 rows( )和 cols( )分别返回矩阵的行、列数。

4. 调用矩阵 Matrix<LIB>的函数

Matrix<LIB>的函数具有与 MATLAB 中函数类似的形式。设 a, b 为两个矩阵，要求解 x 满足线性系统 ax=b，可以用下面的语句实现：

```
x=mldivide(a, b);
display(x);
```

mldivide( )函数用来求两个矩阵的商，而 display( )函数显示矩阵元素。

8.3 Matrix<LIB>函数使用参考

下面对 Matrix<LIB>中常见的一些函数进行简单介绍。更详细的信息可以查看 Matrix<LIB>

参考手册或联机文档。

8.3.1 矩阵操作

1. 生成矩阵

Matrix<LIB>中矩阵的类型称为 Mm，常用来定义矩阵变量，但它定义的矩阵是空矩阵。以下几个函数常用于对矩阵赋值：

```
Mm a;                    //定义矩阵变量 a
a=zeros(m, n);           //定义一个 m×n 维的零矩阵
a=zeros(m, n, p);        //定义一个多维的零矩阵
a=czeros(isc, m, n);     //当 isc=true 时，定义一个复数元素的矩阵
```

2. 从一个数生成矩阵

数可以自动地转换成矩阵，如：

```
Mm x;
x=2.7e-3;
    disp(x);
```

则代码的输出为：0.0027。

3. 命名矩阵

矩阵可以在创建时命名，这时它可以用函数 display( )显示，并可以通过 getname( )成员函数访问。矩阵名很方便，但并不是任何时候都使用，如从一个数自动转换得到的不会被命名。

矩阵命名的构造符是 dMm（它实际上是一个宏）。例如下列语句：

```
dMm (a);
a=1.5;
display(a);
```

将会生成如下结果：

```
a (1x1)=1 double elements real (8 bytes) =
1.5
```

与之对应，通过下列语句

```
Mm a;
A=1.5;
display(a);
```

将会生成如下结果：

```
ans (1x1)=1 double elements real (8 bytes)=
1.5
```

如果将上面的函数 display(a)改为 disp(a)，则二者没有差别，显示结果都为 1.5。

此外，由于 dMm 是一个宏，不能同进创建几个矩阵。如果要创建几个矩阵，必须分开，即一次只能创建一个矩阵。

```
dMm(a); dMm(b); dMm(c);          //正确，一次创建一个矩阵
```

如果写成 dMm(a, b) 则编译时会出现错误信息：warning C4002: too many actual parameters for macro 'dMm'。

4. 生成字符串矩阵

转换函数 TM( )提供了将 C 字符串转换为矩阵的能力，例如：

```
Mm x;
x=TM("This sentence is false");
```

代码将会创建一个名字为 x 的矩阵，它的内容是上面的句子。用 display(x)命令显示结果如下：

```
ans (1x22)=22 char elements    (176 bytes) =
This sentence is false
```

成员函数 setstr(int)可以用来设置或者改变矩阵的字符串属性（stringness），它主要是让 display( )函数知道怎样显示矩阵，如：

```
Mm x;
x=TM("This sentence is false");
x.setstr(1);
 disp(x);
```

则会显示"This sentence is false"，而一些说明信息没有了。

5. 转换字符串矩阵为指向字符的指针

与 TM( )相反，函数 Mstr( )提供了将字符串矩阵转换为 char *类型的能力，它的原型是：

```
void DLLI Mstr(cMm x, char *str, int maxlen);
```

其中，x 是源矩阵，str 是指向字符串的变量，而 maxlen 指定可以存储在 str 中的字符的最大数目。当矩阵中包含的字符长度超过 maxlen 时，后面的部分将会被截尾。

6. 访问矩阵的内容

在 C++中访问（读或写）矩阵元素是通过两个成员函数进行的。设 x 为一个矩阵，则可以通过 x.r( ) 访问矩阵的实部；x.i( ) 访问矩阵的虚部。

当使用这两个成员函数时，它会检查参数是否在矩阵的维数之内。如要不是，将会出现错误提示。根据需要，它有以下 4 种具体形式，以.r( )为例：

- x.r( )　　　返回矩阵(1, 1)位置的值；
- x.r(n)　　　返回矩阵(1, n)的值；
- x.r(i, j)　　返回矩阵(i, j)位置的值；
- x.r(i, j, k)　多维矩阵返回矩阵(i, j, k)位置的值。

对.i( )的使用也是类似的。

```
Mm x;
double a, b, c;
x=magic(3);   //x=[ 8 1 6; 3 5 7; 4 9 2]
a=x.r( );     //a=8
b=x.r(2);     //b=3
```

```
c=x.r(3,2);     //c=9
x.r( )=-x.r( );   //x=[-8, 1 6; 3 5 7; 4 9 2]
```

7. 直接访问矩阵的内存

当许多矩阵元素需要读、写时，可以直接使用矩阵在内存中的地址访问。它同样是通过两个成员函数实现的。

- x.addr( ) 返回指向矩阵实部的指针；
- x.addi( ) 返回指向矩阵虚部的指针。

类似于.r( )，它们也根据需要有三种具体的调用形式：

- x.addr( )　指向矩阵元素(1, 1)的指针；
- x.addr(n)　指向矩阵元素(1, n)的指针；
- x.addr(i, j) 指向矩阵元素(i, j)的指针。

现举例说明如下：

```
Mm x;
double a, b, c;
x=magic(3);    //x=[ 8 1 6; 3 5 7; 4 9 2]
a=*x.addr( );    //a=8
b=*x.addr(2)   //b=1;
*x.addr(3, 1)= -999;     // x=[ 8 1 6; 3 5 7; -999 9 2]
*x.addr( )=- -x.addr( );   // x=[ -8 1 6; 3 5 7; -999 9 2]
```

这里.addr( )也可以用来一次将矩阵的所有元素读到一指向双精度数的指针中，以便与其他程序接口。

```
double a_v=(double* ) malloc(512*sizeof(double));
Mm a=some_function( );    //假设 a 是一个 512×1 的矩阵
memcpy(a_v, a.addr( ), 512*sizeof(double));
```

8. 从双精度常数向量生成矩阵

可以通过宏 M_VECTOR( )从一个双精度数的矢量初始化化矩阵。例如：

```
double a_v={ 1, 3, 5.4, 0, -1, 4.6};
Mm a;
M_VECTOR(a, a_v);
```

然后可以通过 reshape( )函数将收到的向量改为自定义维数矩阵。

```
reshape(2, 3);   // 将 a 改为 2×3 的矩阵
```

9. 从指向双精度的指针中初始化矩阵

从一个非零的双精度向量初始化一个矩阵，可以使用.r( )成员函数逐个初始化，也可以按如下方法完成：

```
double *a_v;      //假设 a_v 是一个已定义的指向双精度常数块的指针
Mm a=zeros(1, 10);      //定义矩阵向量
memcpy(a.addr( ), a_v, 10*sizeof(double));     //初始化矩阵
```

10. 从一列数中生成矩阵

BR( )函数可以从一列数中生成矩阵，它易于使用。例如：

```
Mm a=((BR(1), 0, -1, semi, i, pi, 5, semi, 0.3,-4, 0);   display(a);
```

则生成了一个复数矩阵变量：

```
ans (3x3)=9 double elements complex (72 bytes) =
   1          0          -1
 0+1i       3.142          5
0.3          -4          0
```

其中，semi 相当于 MATLAB 矩阵变量赋值中的分号。

11. 一些矩阵类的成员函数

下面介绍一些矩阵类的成员函数。假设 x 是一个矩阵变量，所有的矩阵成员函数都可以用 x.func( )访问。

.size( )　返回矩阵元素的个数；

.size(dim)返回第 dim 维的元素个数；

.rows( )　返回矩阵的行数；

.cols( )　返回矩阵的列数；

.isstr( )　判断矩阵是否是字符串矩阵，如果是则返回"1"；

.isc( )　判断矩阵是否是复矩阵，如果是则返回"1"。

8.3.2 库常量

- 虚数单位 i, j；
- Pi　常量 pi=3.1415926；
- Inf　无穷大；
- NaN　不定数，如 0/0；
- eps　最小的数(定义为 2^-52)，1+eps<>1；
- semi　常与 BR( )一起用于分隔矩阵行，例如 a=(BR(1), 2, semi, 3, 4)；
- c_p 相当于 MATLAB 中的 colon(:)，例如：

```
Mm a3=(BR(1), 0, -1, semi, i, pi, 5, semi, 0.3, -4, 0);
Mm b2=a3(c_p,2);   display(b2);
```

返回的结果为：

```
ans (3x1)=3 double elements complex (24 bytes) =
  0
 3.142
-4
```

i_o 作为多返回值的函数调用时的分隔符（placeholder），例如：

Mm x, u, s, v; svd(x, I_o, u, s, v) 会返回三个值 u, s, v。

8.3.3 访问库函数

1. initM( )和 exitM( )

在使用 Matrix<LIB>中的库函数前，必须调用 initM( )以初始化应用程序。结束应用程序前，使用 exitM( )以结束库调用。调用形式为：

```
initM(Matcom_VERSION);  //
......                              //应用程序代码
exitM( );
```

其中，Matcom_VERSION 是一个在 matlib.h 中定义的常量，以确保动态链接库与 matlib.h 对应。

2. 函数参考使用方法

在 Mideva 安装目录的 doc 子目录下，有个独立的名称为 refguide.pdf 的文件，里面有函数的使用方法，如功能描述、输入/输出参数个数和类型等。

例如，MATLAB 中的函数原型为：function y=det(x)。

它在 Matrix<LIB>中的函数原型为：Mm det(cMm x)。

可以按下列形式调用：

```
Mm x=magic(5), y;
y=det(x);
display(y);
```

3. 多输出函数调用

设 rth( )为一个多输出函数。一般情况下，多输出函数时 MATLAB 函数原型和 C++头文件的函数原型为：

```
%m-file style used in the Reference Guide function [r, theta]=rth(x)
function [r, theta]=rth(x);

//C++ header
Mm rth(Mm x, i_o, Mm& r, Mm& theta);

// C++ calling convention
rth(x, i_o, r, theta);
```

以计算矩阵的特征根的函数 eig( )为例，它在 C++工程中的调用形式为：

```
Mm t, v, d;
x=wilkinson(7);
eig(t, i_o, v, d);
disp(TM("eigenvectors are:"));
display(v);
disp(TM("eigenvalues are:"));
display(d);
```

4. 可变参数个数的函数调用规则

一些函数允许可变参数个数的函数调用，原型中一般以 varargin 作为最后一个输入参数。为了调用此类函数，所有的输入参数必须用 CL。例如，plot( )函数的原型为：

```
Mm DLLI plot(cMm varagin);
```

在 C++中，调用形式为：

```
Mm t2; t2=linspace(0, pi);
plot((CL(t2), sin(t2), TM("*")));
```

8.3.4 矩阵 I/O

Matrix<LIB>提供了多种方式执行矩阵 I/O 操作。它们是函数 load( )和 save( )、重载的"stdio.h"中的函数。此外，还有一些特征以方便按希望的格式显示一个矩阵。

1. load( )和 save( )

load( )和 save( )函数分别用来从二值的矩阵文件中读入或向二值的矩阵文件存储。load( )和 save( )函数的语法为：

```
save(filename, (CL(mat0), mat1, ...));
load(TM("data0"), (CL(mat0), mat1, ...));
```

可见，二者都是可变输入参数的函数，也就是对需要向 MAT 文件中存储或者从 MAT 文件中读取的变量数目没有限制。其中，filename 是一个包含字符串矩阵的矩阵变量。如果它是一个指向字符串的指针，则需要用 TM( ) 进行转换，例如：

```
save(TM("data0"), (CL(mat0), mat1, ...));
```

2. 文件输入/输出例程

"stdio.h"中的 I/O 函数被重载以便接收矩阵变量。下面通过一个例子来说明，它将 a 的值写入文件中。

```
Mm c;
c=rand(1,8);
fopen(TM("list"), TM("w"));
fprintf(fid, TM("%g\n"),(CL(a)));
  fclose(fid);
```

3. 按格式显示矩阵变量

有几个函数用于显示矩阵变量：disp(x)显示 x 的值，而 display(x)显示 x 的值和大小、类型等信息。格式化函数 format( )可以指定显示十进制数字时位数，默认值是 format(TM("short"))显示 4 位数字，而 format(TM("long"))则显示 12 位数字。

4. .r( )和.i( )

可以通过.r( )和.i( )逐个地从矩阵中提取出元素，然后单独显示。

8.3.5 图形函数

Matrix<LIB>提供了丰富的图像和图形函数，请参见 Matrix<LIB>参考手册中的以下几个部分：

- 二维图形绘制；
- 三维图形绘制；
- 定制绘图函数 plot( )；
- 色彩函数；
- 用户接口对话框；
- 图像函数句柄。

它们都可以在 MFC 工程中使用：生成独立的可执行文件或者动态链接库。

1. 图形函数调用

许多图形函数都接收可变数目的输入参数，因此常用到 CL( )。详见前文"可变参数个数的函数调用规则"。

2. 窗口绘图

可以将图形显示定位到一个已经存在的窗口中，这时需调用 winaxes( )函数。它的原型是：

```
Mm DLLI winaxes(void *winparent);
```

其中，winparent 为窗口句柄，返回值为 Mm 类型的变量。例如：

```
Mm plothandle = winaxes(m_pMainWnd->m_hWnd);   //在 MFC 窗口句柄创建坐标轴
```

为了在窗口中定位图形，可以使用 RealPosition 属性。它接收一个 4 元素的向量[left top width height]，这 4 个值都以像素为单位。

```
Mm pos;
Mm plothandle=winaxes(m_hWnd);
pos=(BR(100), 100, 400, 200);
set(plothandle, TM("RealPosition"), pos);
```

当窗口中只有一幅图形时，可以简单地通过 axisposition( )函数，而不用设置 RealPosition 属性。

```
Mm pos;
Mm plothandle= winaxes(m_hWnd);
axesposition(10,10,400,200);
```

也可以向同一个窗口绘制多个图形，例如：

```
Mm pos;
Mm h1=winaxes(m_hWnd);
axesposition(20, 20, 200, 200);
contour((CL(hadamard(16))));

Mm h2=winaxes(m_hWnd);
axesposition(240, 240, 200, 200);
  plot((CL(magic(50))));
```

3. 隐藏图形菜单

当不是向 Windows 窗口而是向图形自己的窗口，即没有使用 winaxes( )函数时，图形会出现自己的“Figure”菜单，如图 8-1 所示。

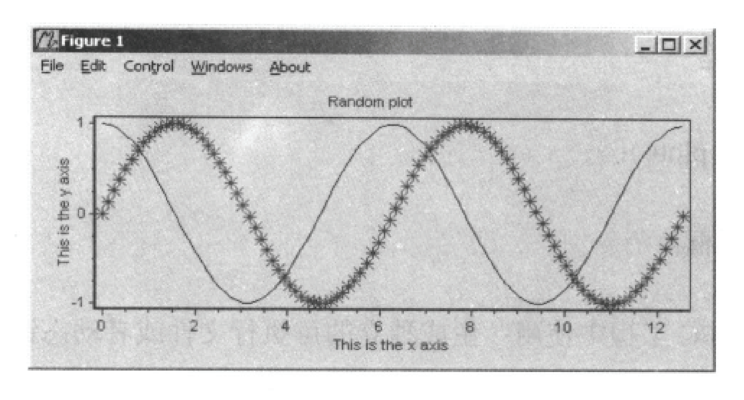


图 8-1　带有菜单的窗口绘图

为了隐藏窗口菜单，可以设置图形的 MenuBar 属性为“None”。为了隐藏 About 菜单项，可以设置图形的 MenuAbout 属性为“off”，即：

```
set(gcf( ), TM("MenuAbout"), TM("off"));
set(gcf( ), TM("MenuBar"), TM("None"));
```

隐藏了窗口菜单的窗口如图 8-2 所示，它已经去掉了窗口菜单。

4. 改变窗口的图标

每个生成的窗口都有自己的图标，如图 8-1 所示，它是默认的图标。因为 Matrix<LIB> 实际上是 Mideva/Matcom 的内核，所以默认的图标是 Mideva 应用程序的图标。当然也可以通过设置窗口的 IconFile 属性来改变图形窗口的图标。

通过 Windows 搜索 MATLAB 的图标文件 MATLAB.ico，并将它复制到应用程序所在的目录，然后设置 IconFile 属性：

```
set(gcf( ), TM("IconFile"), TM("MATLAB.ico"));
```

改变了窗口图标的窗口如图 8-2 所示，窗口图标已经改为 MATLAB 的图标。

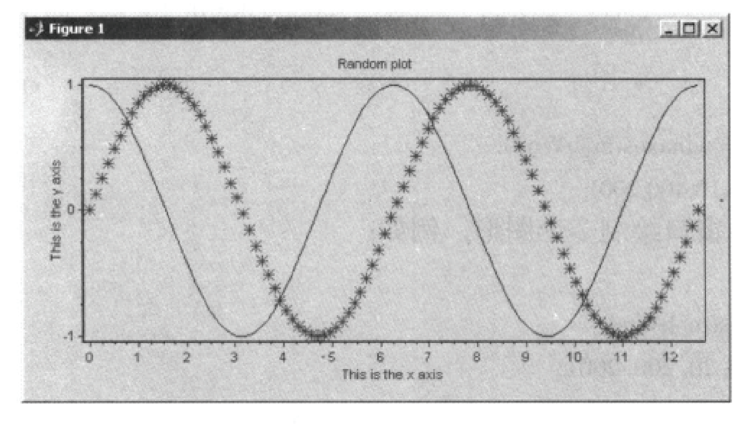


图 8-2　隐藏了菜单的窗口绘图

5. 强制绘图函数

默认情况下，所有的图形输出时会被缓冲而不是立即输出。如果需要立即绘图，可以使

用强制绘图函数 drawnow( )。

8.4　混合编程实例

为介绍 Matrix<LIB>在 Visual C++工程的应用，下面通过两个具体的实例来进行说明。一个是 Win32 Console Application，另一个是利用 MFC 的多文档应用（MDI）应用程序。它们涵盖了本章介绍的绝大部分内容。

【例 8-1】 利用 Visual C++建立一个 Win32 Console Application。它主要用来求解如下线性系统方程的解，并穿插了本章的一些细节内容。

$$\begin{cases} 3a+b+7c=7 \\ 11a+4b+6c=3 \\ 9a+2b=-20 \end{cases}$$

具体包含如下步骤：

（1）启动 Visual C++，建立一个新工程，选择 Win32 Console Application，工程名为 MatrixLIB。在“Step 1 of 1”选择“A simple application”，其余各项采用默认设置。

（2）使用库的头文件 matlib.h。在 MatrixLIB.cpp 文件中加入#include <matlib.h>，并同时告诉编译器文件 matlib.h 所在的路径，即通过向菜单项“Project / Settings / C, C++ / Preprocessor / Additional include subdirectories”添加“c:\matcom45\lib”来完成此项工作。

（3）添加库文件。在 Visual C++中利用菜单命令 “Project/Add to Project / Files ”将 v4500v.lib 加入到 C++工程中。v4500v.lib 位于 c:\matcom45\lib 目录下。

（4）添加必须的源代码。在 main( )函数中添加以下代码：

```
// MatrixLIB.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"

#include <matlib.h>

int main(int argc, char* argv[])
{
  initM(Matcom_VERSION);       //初始化类库

  Mm a;  //初始化矩阵变量
  Mm b;
  Mm x;
  a=zeros(3,3);      //给矩阵变量赋初值
  a.r(1,1)=3;   a.r(1,2)=1;   a.r(1,3)=7;
  a.r(2,1)=11; a.r(2,2)=4;   a.r(2,3)=6;
  a.r(3,1)=9;   a.r(3,2)=2;   a.r(3,3)=0;
  b=(BR(7),semi, 3, semi, -20);

  display(a);
```

```
printf("\n");
display(b);

x=mldivide(a,b);    //求解线性系统方程
//    display(x);
//    printf("\n");

for (int i=1;i<=x.rows( );i++)      //显示解向量
{
      for (int j=1;j<=x.cols( );j++)
      {
            printf("x(%d,%d) = %14.6g",i,j,x.r(i,j));
      }
      printf("\n");
}

double *a_v=(double* ) malloc(5*sizeof(double));
Mm a2=(BR(1),1, 3, 5.4, 0, -1);    //假设 a 是一个 512×1 的矩阵
memcpy(a_v, a2.addr( ), 512*sizeof(double));

//以下请注意 dMm( )和 Mm( )定义变量的细微差别
dMm (y);
y=1.5;
display(y);

Mm (y2);
y2=1.5;
display(y2);

//将字符串转换为字符串矩阵
Mm x2;
x2=TM("This sentence is false");
x2.setstr(1);
disp(x2);

//库常量用法
Mm a3=(BR(1), 0, -1, semi, i, pi, 5, semi, 0.3, -4, 0);
display(a3);
int *w=a3.getdims( );
   disp(*w);
Mm b2=a3(c_p,2);;
display(b2);
```

```
  //绘图
  Mm t2; t2=linspace(0, 2*pi);
  plot((CL(t2), sin(t2), TM("*")));

  exitM( ); //退出 Matrix<LIB>类库

  return 0;
}
```

按 F7 键进行编译后，按 F5 键运行。运行结果如图 8-3 和图 8-4 所示。

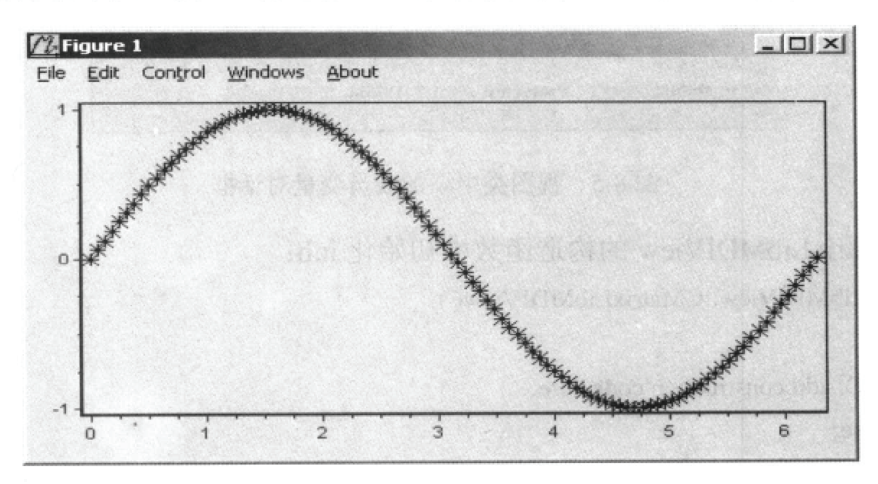


图 8-3 运行结果一

```
"E:\VCTEST\MatrixLIB\Debug\MatrixLIB.exe"
ans (3x3)=9 double elements real (72 bytes) =
 3   1   7
11   4   6
 9   2   0

ans (3x1)=3 double elements real (24 bytes) =
  7
  3
-20
x(1,1) =         -4.975
x(2,1) =        12.3875
x(3,1) =         1.3625
y (1x1)=1 double elements real (8 bytes) =
1.5
ans (1x1)=1 double elements real (8 bytes) =
1.5
This sentence is false
ans (3x3)=9 double elements real (72 bytes) =
  1      0     -1
  4  3.142      5
0.3     -4      0
3
ans (3x1)=3 double elements real (24 bytes) =
    0
3.142
   -4
```

图 8-4 运行结果二

【例 8-2】 利用 Visual C++建立一个基于 MFC 的多文档应用程序。具体步骤如下：

（1）启动 Visual C++，建立一个新工程，选择 MFC AppWizard(exe)，工程名为

MatrixLibMDI。在“Step 1 of 1”选择“Multiple Documents”，其余各项采用默认设置。

（2）使用库的头文件 matlib.h。按例 8-1 的第 2 步进行相同设置。

（3）添加库文件。按例 8-1 的第 3 步进行相同设置。

（4）添加必须的源代码。

① 在视图中添加 bool 类型的 public 成员变量 init，方法是在 Workspace 中单击 ClassView 选项卡，选中 CMatrixLibMDIView，同时单击右键，选择“Add Member Variable”，出现添加成员变量对话框，如图 8-5 所示。

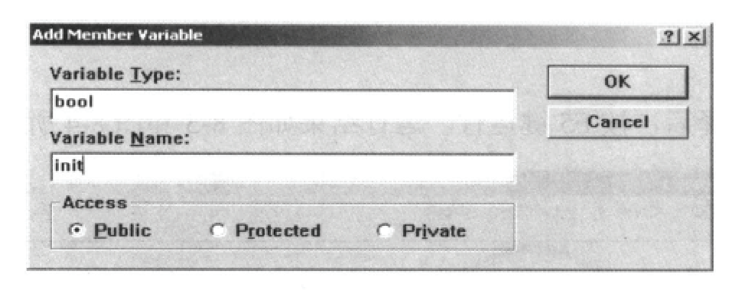


图 8-5　视图类中添加成员变量对话框

② 在 CMatrixLibMDIView 的构造函数中初始化 init：

```
CMatrixLibMDIView::CMatrixLibMDIView( )
{
 // TODO: add construction code here
 init=false;
}
```

③ 通过 Windows 搜索 MATLAB 的图标文件 MATLAB.ico，并将它复制到应用程序所在的目录。

④ 在 Ondraw( )函数中添加如下代码：

```
void CMatrixLibMDIView::OnDraw(CDC* pDC)
{
 CMatrixLibMDIDoc* pDoc = GetDocument( );
 ASSERT_VALID(pDoc);
 // TODO: add draw code for native data here

 if (!init)
 {
      initM(Matcom_VERSION);
      Mm t;
      t=linspace(0,4*pi);
      plot((CL(t),sin(t),TM("r*")));
      hold(TM("on"));      //准备画余弦曲线
      plot((CL(t),cos(t),TM("b-")));             //指定画线的颜色
      title((CL(TM("Random plot"))));
      xlabel((CL(TM("This is the x axis"))));    //标注坐标轴
      ylabel((CL(TM("This is the y axis"))));
```

```
        set(gcf( ),(CL(TM("Color")),TM("white")));
        set(gcf( ),(CL(TM("Box")),TM("on")));
        set(gcf( ), TM("MenuAbout"), TM("off"));
        set(gcf( ), TM("MenuBar"), TM("None"));
//set(gcf( ), TM("IconFile"), TM("MATLAB.ico"));

        init=true;
    }
    drawnow( );                //开始画图
    exitM( );
}
```

编译后运行，会出现如图 8-6 所示的结果。注意：图形窗口的图标已经转换为 MATLAB 的图标。

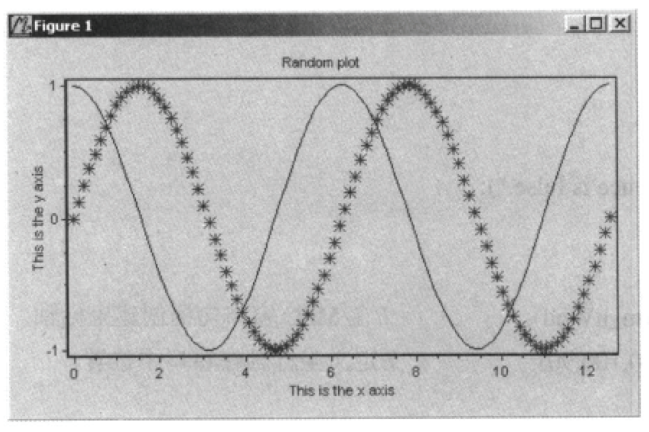


图 8-6　运行结果

有时，应用程序需要将生成的图形在 Windows 的客户区显示。此时，可以通过对 Ondraw( ) 函数中的代码进行简单修改，即增加以下两条语句：

```
Mm h=winaxes(m_hWnd);              //在 MFC 窗口句柄创建坐标轴
axesposition(10, 10, 100, 90);     //定义坐标轴在窗口中的位置
```

修改后的整个 Ondraw( )函数列在下面。此时，重新编译后运行，实验结果如图 8-7 所示。

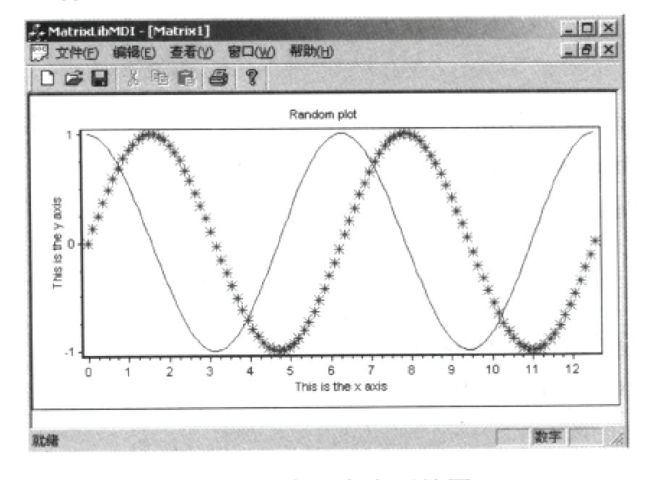


图 8-7　窗口客户区绘图

```
void CMatrixLibMDIView::OnDraw(CDC* pDC)
{
  CMatrixLibMDIDoc* pDoc = GetDocument( );
  ASSERT_VALID(pDoc);
  // TODO: add draw code for native data here

  if (!init)
  {
  initM(Matcom_VERSION);   //Matrix<LIB>初始化

  Mm a, b;
  a=rand(3); b=zeros(3,1);
  b.r(1,1)=3; b.r(2,1)=-1; b.r(3,1)=5;
  display(a);
  display(b);

  Mm x;
  x=TM("This sentence is false.");
  display(x);

  Mm h = winaxes(m_hWnd);               //在 MFC 窗口句柄创建坐标轴
  axesposition(10,10,100,90);           //定义坐标轴在窗口中位置

  Mm   t;
  t=linspace(0, 4*pi);
  plot((CL(t),sin(t),TM("r*")));        //画正弦曲线
  hold(TM("on"));                       //避免屏幕重绘操作

  //准备画余弦曲线
  plot((CL(t),cos(t),TM("b-")));

  //标注坐标轴
  title((CL(TM("Random plot"))));
  xlabel((CL(TM("This is the x axis"))));
  ylabel((CL(TM("This is the y axis"))));

  //设置绘图区属性
  set(h,(CL(TM("Color")),TM("white")));
  set(h,(CL(TM("Box")),TM("on")));

  init=true;
  }
```

```
    drawnow( );            //开始画图
    exitM( );              //退出 Matrix<LIB>
  }
```

8.5 MATLAB 数学库

8.5.1 简介

除了 MathTools 公司的 Mideva 提供了数学库 Matrix<LIB>，可以生成脱离 MATLAB 环境的独立应用程序外，MATLAB 自带了 C++的数学和图形库。MATLAB 的数学库包含了 400 多个常用的 MATLAB 函数，并且其使用习惯和 MATLAB 函数的使用习惯极其相似。对于 MATLAB 的使用者而言，采用 MATLAB 自带的 C++数学库，也可以使应用程序完全脱离 MATLAB 的解释环境。对于 C++的程序开发者，采用 C++的数学库则可以利用已有的矩阵运算的数学库函数，加快程序的开发进度。此外，与 MATLAB 引擎应用程序相比，除了可完全脱离 MATLAB 环境运行外，调用 MATLAB 的数学库和图形库还具有执行速度快和内存需求小的优势。当然，它也具有以下缺点：

- 不能调用图形句柄系统的函数；
- 不能调用 MATLAB 工具箱的函数；
- 用户不能够在基于 MATLAB 数学库和图形库的应用程序中使用 MATLAB 的函数 eval( )和 input( )；
- MATLAB 中的一些方法在 C/C++中不提供支持，例如“；”和“[ ]”。

MATLAB 提供的 C++数学库放在 MATLAB 根目录的 extern 子目录下，其中 include 目录包含了开发 MATLAB C++数学函数库应用程序所必需的头文件和 Visual C++编译所必需的定义文件，而 include 下的 cpp 子目录则包含了一些建立独立可执行的 C++应用程序所必需的头文件。这些头文件和定义文件归纳见表 8-1。

表 8-1 MATLAB 自带的 C++数学库头文件和定义文件

文 件 名	说 明
libmatlb.h	包含了所有 MATLAB 内建数学函数 C 语言实现的函数定义
libmmfile.h	包含了 MATLAB 以 M 文件形式提供的数学函数 C 语言实现的函数定义
matlab.h	MATLAB 的 C 数学函数库的头文件
matrix.h	定义了 mxArrary 结构体以及相应的访问函数
libmat.lib	包含了从 MAT 文件动态链接库中输出的函数名，用于 Visual C++
libmatlb.lib	包含了从 MATLAB 内建数学函数动态链接库输出的函数名，可用于 Visual C++编译器
libmmfile.lib	包含了从 MATLAB 以 M 文件形式数学函数动态链接库中输出的函数名，可用于 Visual C++
libmx.lib	包含了从动态连接库 libmx.dll 中输出的函数名，可用于 Visual C++
matlab.hpp	包含 C++应用程序必须包含的头文件
Version.h	包含了 C++编译器的信息
mathwork.h	包含数值变量的声明

值得说明的是，MATLAB 提供的 C++数学库与 MATLAB C 数学库类似，只有 MATLAB 阵列一种数据类型。MATLAB C++数学库中用 mwArray 类封装了 MATLAB 阵列的数据和基本操作函数，而 MATLAB 对矩阵进行操作的函数，MATLAB C++数学库通过提供外部函数的方式供用户调用。

8.5.2 Visual C++工程中调用 MATLAB 数学函数库的环境设置

在 Visual C++集成开发环境下，使用 MATLAB 的 C++数学函数库时必须对集成开发环境进行配置。配置过程与第 6 章通过 MAT 文件实现数据共享时的环境配置相似，首先需要加入静态链接库，然后设置 C/C++选项卡中的选项，并在工程中包含相应的头文件。具体的配置过程如下：

1. 需要加入或忽略的静态链接库

对于使用 MATLAB C++数学库的 Visual C++工程人员来说，Visual C++工程设置中需要加入的静态链接库有：libmatpm.lib、libmx.lib、libmatlb.lib、libmat.lib、libmmfile.lib、sgl.lib 和 libmwsglm.lib。设置方法是在菜单"Project"中选择"Setting"子菜单，在"Link"选项卡中添加上述静态库文件，每个库文件以空格隔开，如图 8-8 所示。

其中，sgl.lib 和 libmwsglm.lib 只有在用到 MATLAB C++图形库的时候才需要在 Visual C++的工程设置中添加这两个静态链接库。另外，在 Visual C++工程中需要忽略 Visual C++的 MSVCRT 静态库。

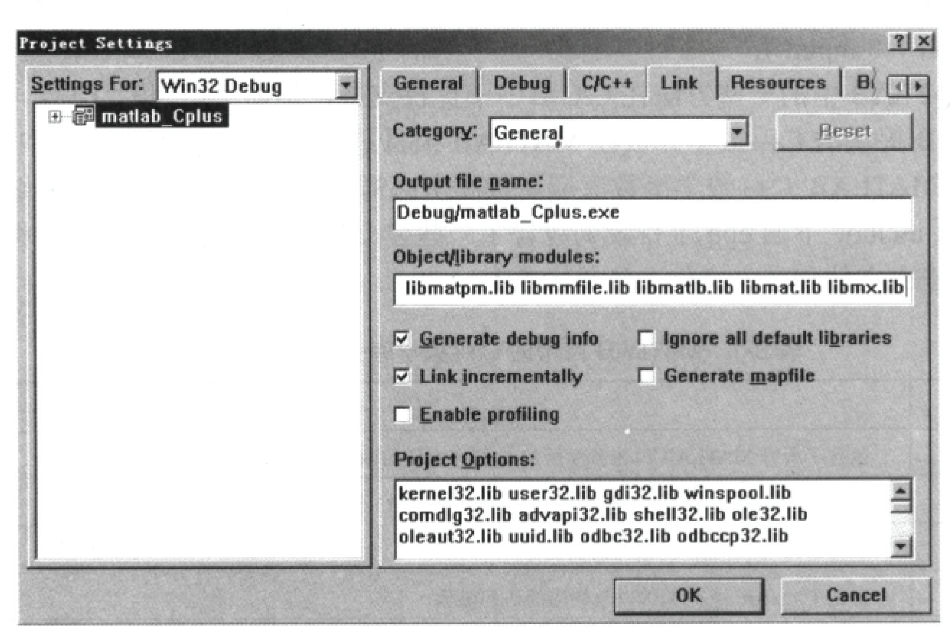


图 8-8 使用 MATLAB C++数学库需要加入的静态链接库

2. 设置 C/C++选项卡的选项

在 Visual C++菜单"Project"中选择"Settings"子菜单，在"C/C++"选项卡的"Category"下拉列表框中选择"Code Generation"，将"Use run-time library"选项设为"Multithreaded DLL"，如图 8-9 所示。

同时，在"Category"列表框中选择"Proprocessor"，并在"Preprocessor definitions"选

项中增加如下内容：MSVC，MSWIND，IBMPC，D__STDC_，_AFXDLL。注意：这几个符号之间用逗号分隔。

3. 在工程中包含相应的头文件

在 Visual C++工程中不用 MATLAB C++图形库的时候，只需包含头文件 matlab.hpp 即可；如果要用到 MATLAB C++图形库，则需要包含头文件 matlab.hpp 和 libmwsglm.h。

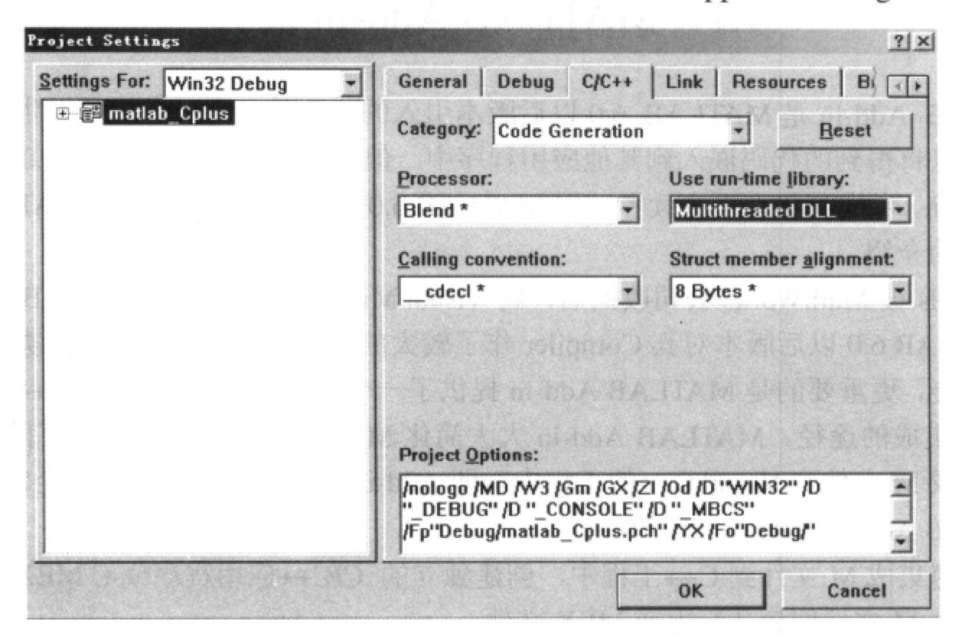


图 8-9 改变 run-time library 的设置

8.6 小　　结

本章介绍了利用 Mideva 工具包含的 Matrix<LIB>数学库与 Visual C++进行混合编程的方法，包括 Matrix<LIB>的安装、语法和 Visual C++集成开发环境的配置，并通过实例说明了如何利用 Matrix<LIB>数学库生成可脱离 MATLAB 运行环境的独立应用程序。对于 MATLAB 自带的 C++数学库，也对其进行了简要介绍。对于二者，可以在工程实际中灵活应用，但建议使用 Matrix<LIB>。

第 9 章　通过 MATLAB Add-in 实现混合编程

9.1　MATLAB Add-in 简介

MATLAB Add-in 是 MATLAB 6.0 以后版本引入的新工具，它可以方便地生成独立应用程序，也可以将得到的代码嵌入到其他应用程序中。但是，最新的 MATLAB 7.0 以后版本不再支持 Add-in。考虑到仍有不少读者使用 MATLAB 6.0 或 6.5 版本。本章对 MATLAB Add-in 仍进行简单的介绍。

MathTools 被 MathWorks 公司收购后，将 Visual Matcom 集成到了 MATLAB 6.0 版本中。因此，MATLAB 6.0 以后版本对其 Compiler 作了较大的改进，支持更多的数据类型，具有更强的优化功能，更重要的是 MATLAB Add-in 提供了一个 MATLAB 和 Visual C++集成开发环境 IDE 直接集成的途径。MATLAB Add-in 大大简化 M 文件在 Visual C++环境下的使用，它可以自动集成 M 文件到 Visual C++的工程中，即 Add-in 与 Visual C++环境是全集成的。

MATLAB Add-in 具有以下新的特征：

- 快速集成 M 文件到 C++工程中，创建独立的 C/C++应用程序或 C MEX DLL；
- 通过 M 文件创建共享库或 MEX 文件；
- 内含 Visual Matrix Viewer，调试过程中可以查看矩阵变量的值；
- 直接修改 M 源文件而不是修改生成的 C/C++文件；
- 方便快捷地打包应用程序等。

但是，请注意 MATLAB Add-in 目前并不提供对 Visual C++ 7.0 版本的支持。

9.2　MATLAB Add-in 安装和在 Visual C++中的环境设置

当安装好 MATLAB 6.5 以后，启动 MATLAB 6.5，然后在 MATLAB 命令行中输入以下命令：

```
>> mex –setup
```

会出现以下提示，它将配置 MEX 和独立应用程序使用 Visual C++为默认的编译器，并安装 MATLAB Add-in 所需 MATLAB Compiler 和 C/C++数学库文件到 MSVisual C++目录。

```
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Digital Visual Fortran version 6.0 in C:\Program Files\Microsoft Visual Studio
[2] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual Studio
```

[0] None

Compiler: 3

输入“3”，选择 Microsoft Visual C++ 6.0 作为外部编译器。然后，系统会出现以下提示，要求验证系统自动搜索到的 Visual C++的路径是否正确。

```
Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0
Location: C:\Program Files\Microsoft Visual Studio

Are these correct?([y]/n):
```

输入“y”进行确认，系统会出现以下提示：

```
Try to update options file: C:\Documents and Settings\Administrator\Application
Data\MathWorks\MATLAB\R14\mexopts.bat
From template: C:\MATLAB7\BIN\WIN32\mexopts\msVisual C++60opts.bat

Done . . .
```

这时，针对 Visual Studio 的 MATLAB Add-in 已自动安装到系统上。然而，还需要进行以下几步，以确保能在 Visual C++中使用 MATLAB Add-in。

在 MATLAB 环境下运行以下命令：

```
>>cd(prefdir);
>>mccsavepath;
```

它将保存当前 MATLAB 路径 mccpath 文件到用户路径 prefdir，因为 MATLAB Add-in 脱离 MATLAB 运行，否则它无法知道 MATLAB 路径。

在 MSVisual C++环境中配置 MATLAB Add-in，方法是：从菜单“Tools -> Customize”选择“Add-ins and Macro Files”选项卡，选中“MATLAB Add-in”，单击“Close”。Visual C++工具栏出现 MATLAB Add-in 图标（如图 9-1 所示），以后每次启动 Visual C++，自动加载 MATLAB Add-in。

其实，当在机器上安装了 Mideva 后，在 Visual C++环境下进行配置，则可以在 Visual C++的工具栏中见到 Visual Matcom 的图标，如图 9-2 所示。具体的配置方法如下：

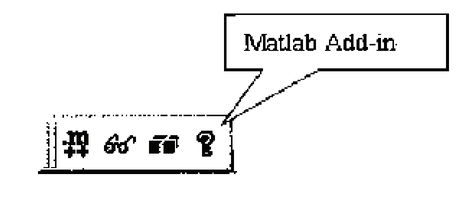


图 9-1 工具栏中 MATLAB Add-in 图标

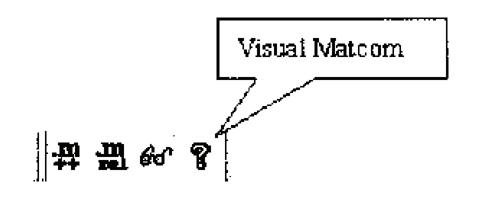


图 9-2 工具栏中 Visual Matcom 图标

（1）启动 Visual C++；

（2）从 Visual C++菜单“Tools”中，选择“Customize”，再选择“Add-in and Macro Files”选项卡。

（3）单击“Browse”，在“Browse for Macro File or Add-in”中将文件类型选择为“Add-ins (.dll)”，将查找范围转换为 C:\Matcom45\bin 目录下，选择 mvcide.dll，然后单击打开按钮（如图 9-3 所示）。

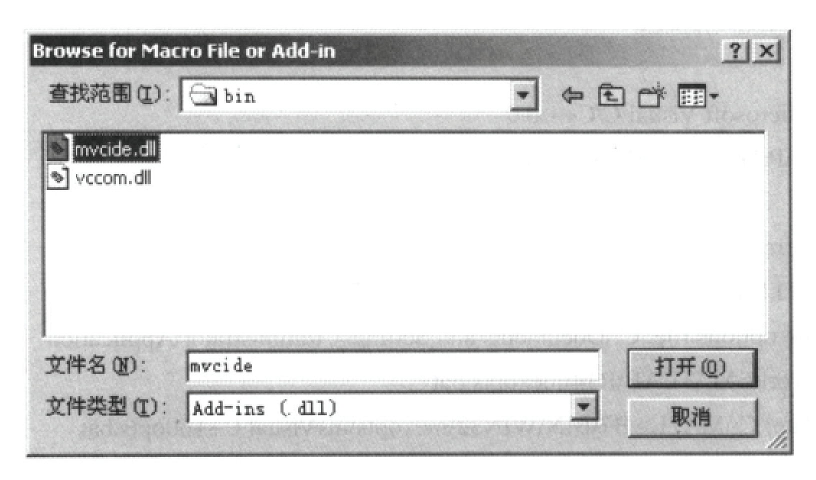


图 9-3　Visual C++工具栏中配置 Visual Matcom 图标对话框之一

（4）出现如图 9-4 所示的对话框，选中“MATLAB Add-in”，然后单击“Close”按钮。

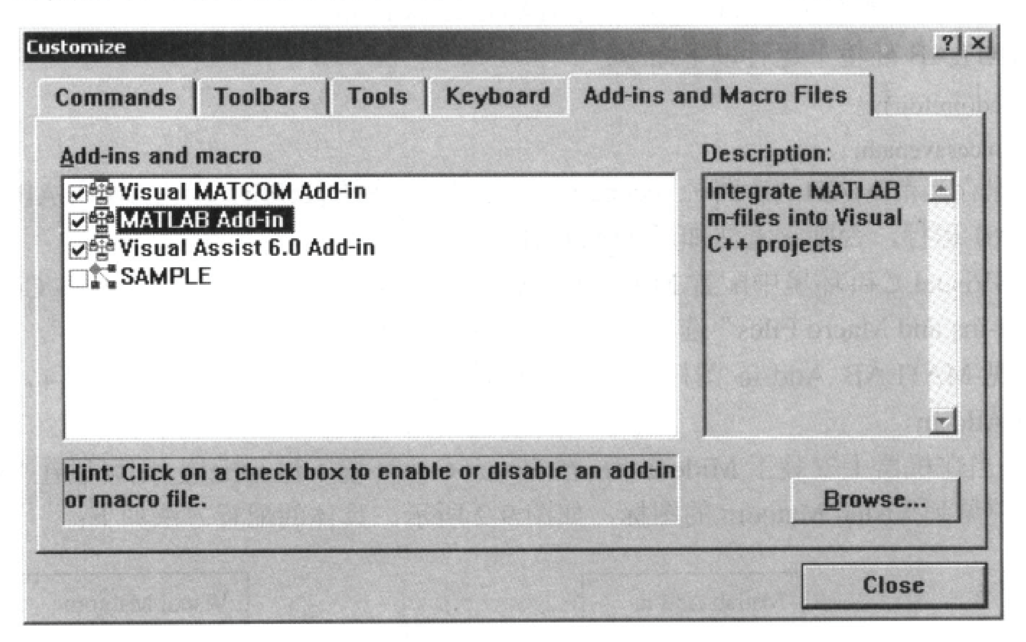


图 9-4　Visual C++工具栏中配置 Visual Matcom 图标对话框之一

很显然，MATLAB Add-in 的图标与 Visual Matcom 的图标很类似，只是增加了打包独立应用程序功能。因此，可以说 MATLAB Add-in 实际上是在 MathWorks 收购 MathTools 后将 Visual Matcom 集成到 Visual Studio 的产物。在后面的实例中将会发现，通过 MATLAB Add-in 和 Matcom 将 M 文件转换成的 C/CPP 代码非常相似，而且与 Matrix<LIB>接近。

当在 Visual C++环境中配置好 MATLAB Add-in 后，如果重新启动 Visual C++建立一个新工程，则在建立新工程的工程向导中多了一个新的应用程序向导 MATLAB Project Wizard

（如图 9-5 所示）。

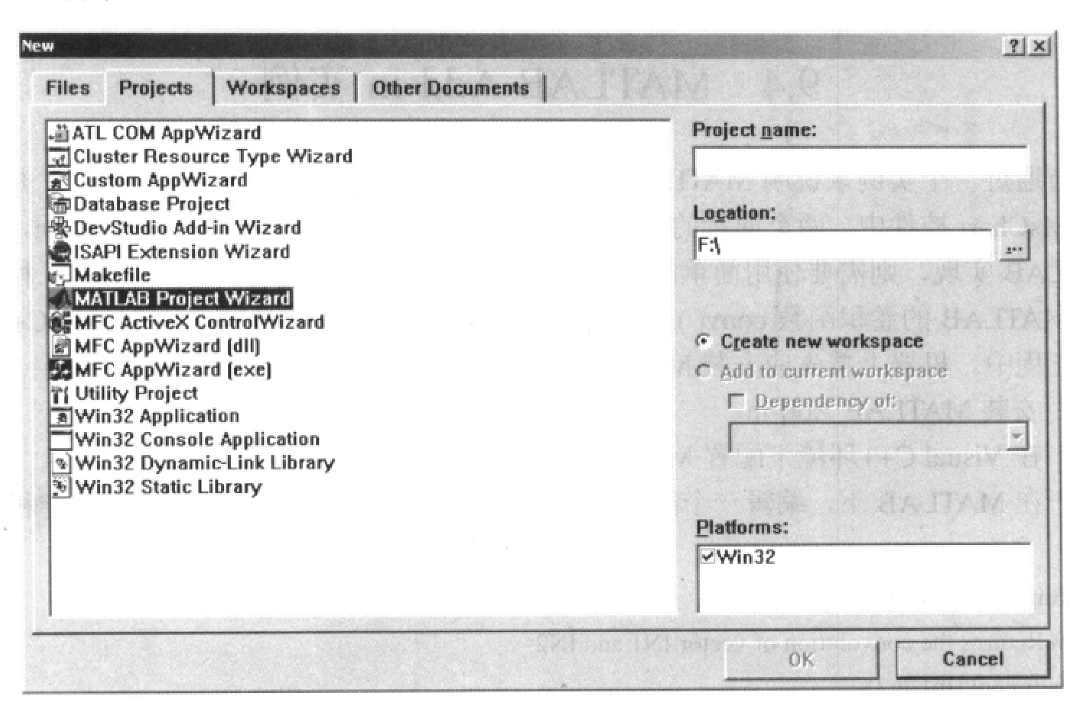


图 9-5　Visual C++中新出现的 MATLAB 工程向导

当需要改变默认的编译器 Visual C++为其他的编译器，如 C++ Builder 时，可以按前面所述在 MATLAB 环境中重新运行 mex –setup。

9.3　通过 MATLAB Add-in 生成独立应用程序

通过 MATLAB Add-in 可以方便地生成独立应用程序，但是在使用过程中需要注意几点：

（1）添加到 MSVisual C++工程的文件应当为一个函数文件，而不能是一个 Script 文件，否则会出现错误提示“a script M-file cannot be compiled with the current Compiler”。将一个 Script 文件转换为一个函数文件的方法是添加函数定义行和必要的注释，其中定义行主要是定义函数的名称，以及输入/输出参数的数量和名称，而注释行则是为了增加函数体中代码的可读性而加入的说明性文字。

（2）最好不要修改生成的 C/C++代码，如果需要改动，可以通过在 MSVisual C++工作区中直接修改 M 文件（无须在 MATLAB 环境中），重新编译即可。可以通过在 M 文件中设置断点，在运行中通过 Matrix Viewer 观察 MATLAB 变量的值。

（3）在 Add-in 所生成的 Readme 中可以看到如下的语句：

```
This is the output from running MATLAB Add-in.Compiler version: 3.0
    mcc -k "G:\Visual C++TEST\TestNew\mcc.mak" -/n -p -A line:on -g libmmfile.mlib -vh
"D:\MATLAB6p5\work\myplot.m"
```

其中，mcc 是 MATLAB 的编译器内核，它提供的是一种命令行的编译方式，可以指定参数。通过 MATLAB Add-in 则可以简化 mcc 的使用，避免复杂的参数设置。

目前通过 Add-in 实现 MATLAB 与 Visual C++的混合编程也有一定的不足，如目前并不支持所有的 MATLAB 的函数；生成的程序代码有些复杂等。估计 MATLAB 6.0 以后的版本

在这方面肯定有所改进的。

9.4 MATLAB Add-in 实例

本节通过一个实例来说明 MATLAB Add-in 的应用。它计算两个向量的卷积，并将结果显示在 MsChart 控件中。两个向量的卷积操作如果采用 C 语言实现则编程比较麻烦，如果采用 MATLAB 实现，则需要使用简单的卷积函数 conv( )。因此，在 Visual C++的工程中，可以利用 MATLAB 的卷积函数 conv( )来实现，通过 MATLAB Add-in 将其转换为 C/C++代码，添加到工程中。机器上首先应安装 MATLAB R13，并支持 C 数学库和 compiler。

（1）安装 MATLAB Add-in。

（2）在 Visual C++环境下配置 MATLAB Add-in。

（3）在 MATLAB 下，编辑一个简单的函数文件 Myfunc.m，实现两个向量的卷积。其代码如下：

```
function out=Myfunc(In1,In2)
%Returns the convolution of vector IN1 and IN2
out=conv(In1,In2);
```

（4）在 Visual C++环境下创建一个 MFC AppWizard (exe)工程，选择生成一个基于对话框的应用程序，命名为 conv。在对话框中添加一个按钮，其 ID 为 IDC_BUTTON1，CAPTION 为 Conv，并通过 MFC ClassWizard 添加按钮对应的左键单击响应事件 CConvDlg::OnConv( )。

（5）单击菜单上的 .m ++ 按钮，按照提示的默认选项（如图 9-6 所示），将 MATLAB 文件 Myfunc.m 添加到工程中。这时 MATLAB Add-in 将自动启动，生成 Myfunc.m 对应的头文件 myfunc.h 以及 C 文件 myfunc.c 和 myfunc_main.c。编译完成后，项目的 Workspace 栏如图 9-7 所示，可以看到添加的 MATLAB M 文件和对应生成的 C/C++文件。

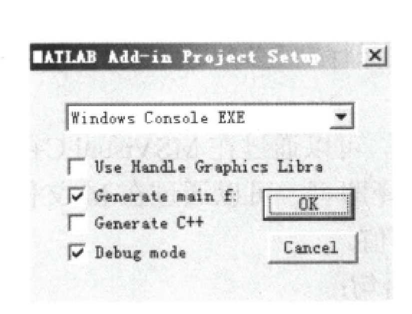


图 9-6　MATLAB Add-in 工程设置对话框

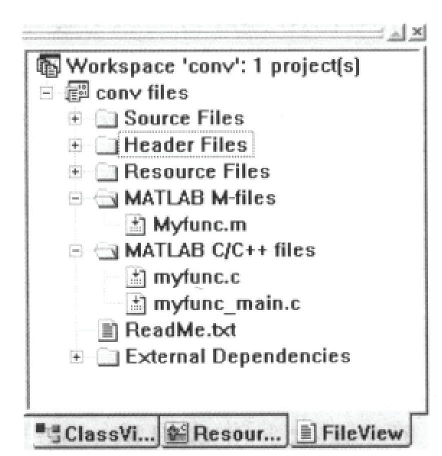


图 9-7　添加 M 文件后的 Workspace 区

这时可以看到工程生成了一个文本文件 mcc-log.txt。可见，MATLAB Add-in 实际上仍然是通过 MCC 完成的。MATLAB Add-in 所用到的编译器批处理文件 compopts.bat，可以在 Windows 的 DOS 模式下通过 type 命令查看，其主要内容都在文本文件 mcc-log.txt 中。文本文件 mcc-log.txt 的主要内容列举如下：

```
This is the output from running MATLAB Add-in.
```

```
--------------MCC/MEX BEGIN---------------
mcc -k "D:\Visual C++test\mcc.mak" -/n -m -A line:on -g libmmfile.mlib -vh "D:\Visual C++test\Myfunc.m"
Compiler version: 3.0
Parsing file "d:\Visual C++test\myfunc.m"
  (Referenced from: "Compiler Command Line").
Generating file "myfunc.h".
Generating file "myfunc.c".
Generating file "myfunc_main.c".
……

-> Default options filename found in C:\Documents and Settings\Yang Gaobo\Application Data\MathWorks\MATLAB\R13
-------------------------------------------------------------
-> Options file= C:\Documents and Settings\Yang Gaobo\Application Data\MathWorks\MATLAB\R13\compopts.bat
……
--------------MCC/MEX DONE----------------
```

打开生成的 myfunc.c 文件，可以看到原始的 M 文件 Myfunc.m 的 Myfunc( )，经过 MATLAB Add-in 转换后，其函数原型为：mxArray * mlfMyfunc(mxArray * In1, mxArray * In2)。因此使用时必须按 mlfMyfunc( )的要求进行接口。

（6）给对话框的按钮添加必要的左键单击响应事件代码 CConvDlg::OnConv( )。为此，先要在 ConvDlg.cpp 文件开头添加必要的头文件，即添加以下两行代码：

```
#include "matlab.h"
#include "Myfunc.h"
```

事件 CConvDlg::OnConv( )完成两个向量的卷积计算，并将结果显示在 MsChart 控件中。以下仅列出与 MATLAB Add-in 相关的向量卷积计算源代码，其余代码请参阅本书光盘中的内容。

```
void CConvDlg::OnConv( )
{
  // TODO: Add your control notification handler code here

  //x1, x2: 参与卷积的两个向量
  //res: 存放卷积的结果向量，其长度为(length(x1)+length(x2)-1)=19
  double x1[]={1,2,3,4,5,6,7,8,9,0};
  double x2[]={4,5,6,7,4,5,8,9,0,7};
  double res[19];

  //按 mlfMyfunc( )的要求进行接口，先生成矩阵向量，初始化
  mxArray* In1;
  mxArray* In2;
```

```
    mxArray* Out;
    In1=mxCreateDoubleMatrix(1,10,mxREAL);
    In2=mxCreateDoubleMatrix(1,10,mxREAL);
    memcpy(mxGetPr(In1),x1,10*sizeof(double));
    memcpy(mxGetPr(In2),x2,10*sizeof(double));

    //调用 mlfMyfunc( )函数进行向量卷积运算
    Out=mlfMyfunc(In1,In2);

    //得到的结果矢量回送给 Visual C++
    memcpy(res,mxGetPr(Out),19*sizeof(double));

    //释放临时变量
    mxDestroyArray(Out);
    mxDestroyArray(In1);
    mxDestroyArray(In2);
  .......    //将卷积结果显示到 Ms Chart 控件，略

  }
```

（7）编译本工程，编译完成后可以看到左侧的 Workspace 区自动地增加了一些 External Dependencies 文件。它们是本工程为支持 MATLAB Add-in 需要的头文件，包括 libmatlb.h、libmatlbm.h、libsgl.h、matrix.h、mex.h 和 matlab.h 等文件。

（8）运行，将出现如图 9-8 所示的界面，显示了原始的两个向量值。单击“CONV”按钮，出现如图 9-9 所示的界面，它同时显示了原始的两个向量及其它们的卷积。

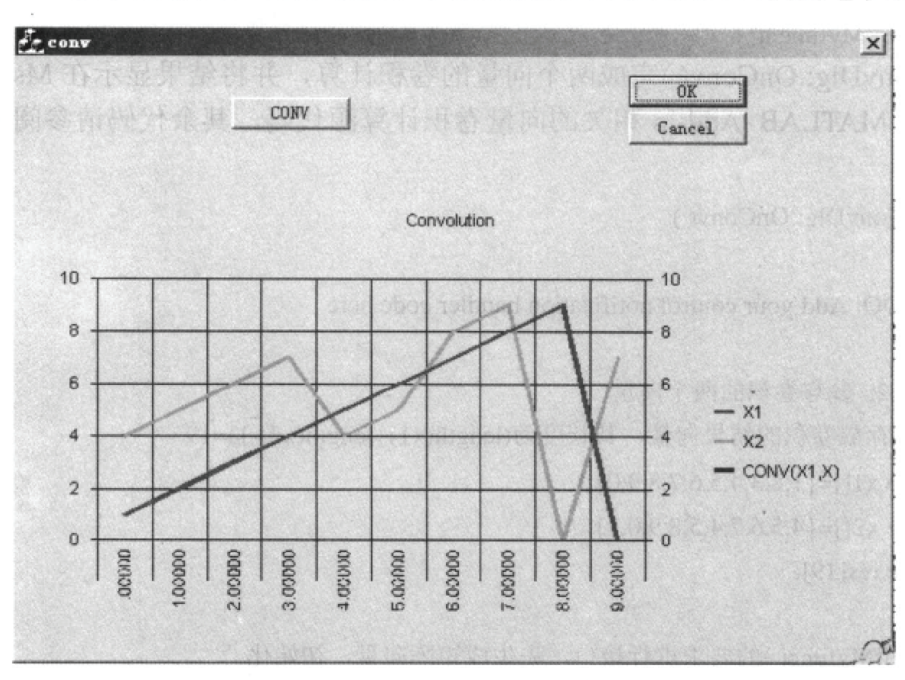


图 9-8　程序运行启动界面，显示原始的两个向量

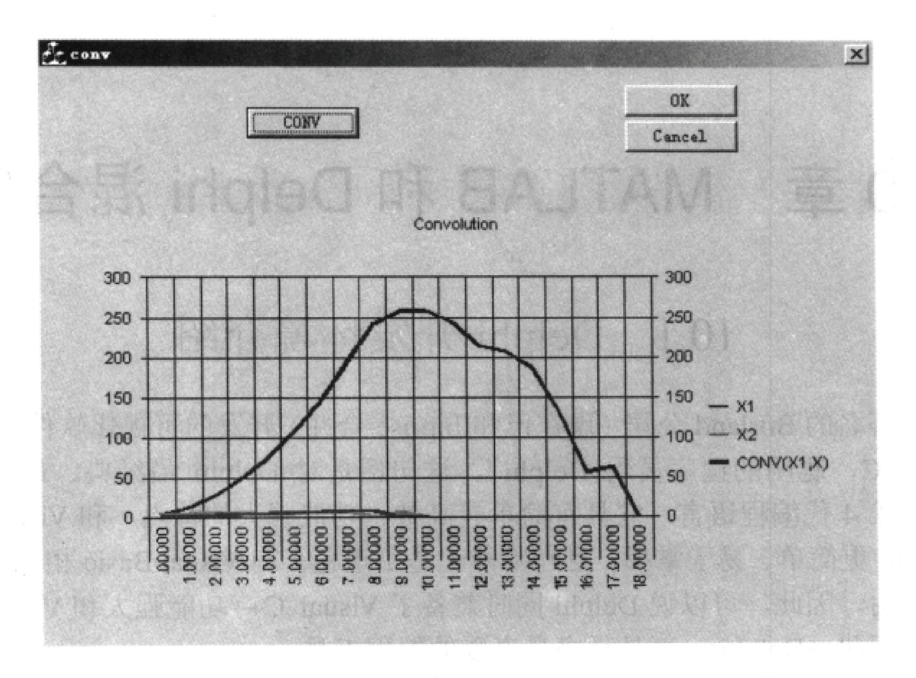


图 9-9　程序运行结果，包含原始的两个向量和它们的卷积

9.5　小　　结

MATLAB Add-in 提供了一种实现 MATLAB 和 Visual C++混合编程的直接集成的途径，通过它可以很方便地生成 MATLAB 和 Visual C++的混合程序。在图像处理等需要大量计算的工程应用场合有广泛的应用价值。

第 10 章　MATLAB 和 Delphi 混合编程

10.1　Delphi 开发环境介绍

Delphi 是著名的 Borland 公司（现在已和 Inprise 合并）开发的可视化软件开发工具。“真正的程序员用 C，聪明的程序员用 Delphi”，这句话是对 Delphi 最经典、最实在的评价。Delphi 被称为第 4 代编程语言，它具有简单、高效、功能强大的特点。和 Visual C++开发环境相比，Delphi 更简单、易于掌握，且在功能上毫不逊色。和 Visual Basic 相比，Delphi 则功能更强大、实用。因此，可以说 Delphi 同时兼备了 Visual C++功能强大和 Visual Basic 简单易学的特点。因此，Delphi 一直是程序员喜爱的编程工具。

Delphi 具有以下的特点：基于窗体和面向对象的方法，高速的编译器，强大的数据库支持，与 Windows 编程紧密结合，强大而成熟的组件技术。Delphi 提供了各种开发工具，包括集成环境、图像编辑（Image Editor），以及各种开发数据库的应用程序，如 Desktop Database 等。此外，还允许用户挂接其他的应用程序开发工具，如 Borland 公司的资源编辑器（Resource Workshop）。

在 Delphi 众多的优势当中，它在数据库方面的特长显得尤为突出：适用于多种数据库结构，从客户机 / 服务器模式到多层数据结构模式；具有高效率的数据库管理系统和新一代更先进的数据库引擎；提供最新的数据分析手段和大量的企业组件。Delphi 通过不断添加和改进各种特性，发展到现在的 Delphi 8.0 版本，功能十分完善、强大。本章以目前广泛使用的 Delphi 6.0 为基础，介绍 MATLAB 和 Delphi 混合编程的方法和技巧。本章的读者对象为具有一定 Delphi 基础的开发者。

10.2　通过 MATLAB 自动化服务实现混合编程

10.2.1　自动化服务的实现方法

自动化服务器是一种可以由其他应用程序驱动的组件，其核心是包含一个或多个供其他应用程序创建和连接的基于 Idispatch 的接口。 MATLAB 作为自动化服务器时，可以被 Windows 平台上任何可作为自动化控制器的应用程序使用。通过使用 MATLAB 自动化服务器功能，用户可以在自己的应用程序中执行 MATLAB 命令，并从 MATLAB 的工作空间中获取数据以及向 MATLAB 输出数据。将 MATLAB 作为一个自动化服务器使用时，必须知道 MATLAB Activex 对象在系统注册表中定义的名字，即 ProgID，一般可以使用 MATLAB.application 或者 MATLAB.application.single，它们代表了不同的含义。当应用程序使用 MATLAB.application 作为 ProgID 启动 MATLAB 自动化服务器时，表示将 MATLAB 自动化服务器作为一个共享的服务器，当其他应用程序以同样的 ProgID 开启 MATLAB 服务器时，系统将不再另外初始化一个服务器，而是使用同一个服务器来完成所有的请求。而当应用程序使用 MATLAB.application.single 作为 ProgID 启动 MATLAB 自动化服务器时，表示将

MATLAB 自动化服务器作为一个单独的服务器，而不与别的应用程序共享。此时，当其他应用程序以同样的 ProgID 开启 MATLAB 服务器时，系统将重新初始化一个服务器。

通常，客户程序获得和控制服务器程序对象的过程如下：

（1）初始化并创建自动化对象，获得服务器程序的句柄，将服务器程序设置为该客户程序的一个服务工具，MATLAB 提供一个自动化对象，它的外部名称为 MATLAB.Application。例如，要将 MATLAB 作为曲线、曲面显示的画板和矩阵运算的工具，为了产生和获得 MATLAB 对象，在 Delphi 中用下列语句实现：

```
MATLAB=CreateOleObject(MATLAB.Application);
```

除了上述开启 MATLAB 自动化服务器的方法之外，还可以在启动 MATLAB 时直接将 MATLAB 初始化为一个自动化服务器，这只需在启动 MATLAB 时使用参数“/automation”即可。一种方法是通过 Winodws 的“开始”菜单下的“运行”子菜单，弹出如图 10-1 所示的对话框，输入“matlab /automation”；另一种方法是通过修改 MATLAB 的桌面快捷方式，右键单击 MATLAB 图标，选择“属性”，将弹出如图 10-2 所示的属性设置对话框，在“目标”后添加“/automation”。

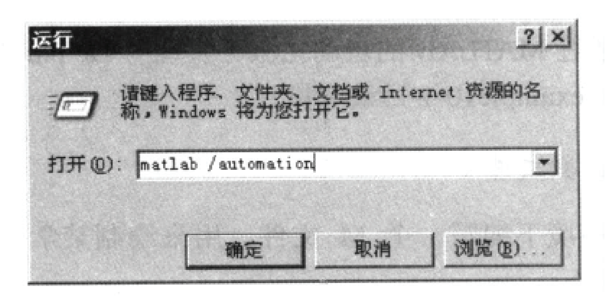


图 10-1　开启 MATLAB 自动化服务器命令

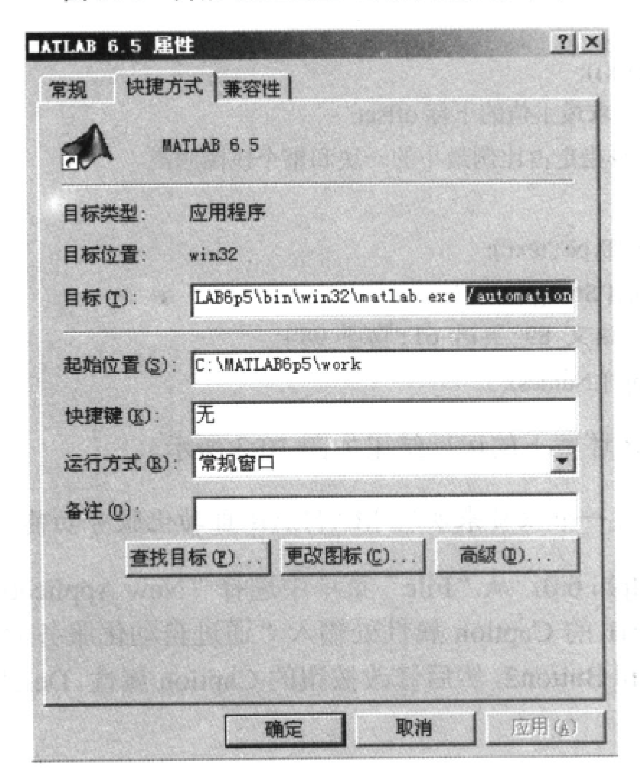


图 10-2　属性设置对话框

（2）编辑和操纵自动化对象的属性和方法。可编辑对象窗口的颜色和大小及成员数据值，使用其成员函数，输入参数而输出经服务器程序处理的信息。例如，获得 MATLAB 的对象后，将其窗口最小化或最大化的程序语句如下：

```
MATLAB.MinimizeCommandWindow();
MATLAB. MaximizeCommandWindow();
```

（3）客户程序释放服务器程序的对象，在 Delphi 中用下列语句编程即可释放 MATLAB 对象：

```
MATLAB.Quit;
```

MATLAB 自动化对象的一个主要函数是 Execute,凡是在 MATLAB 命令窗口可键入的执行命令都可用该函数调用。MATLAB 的自动化对象对数组的获取和设置使用函数 GetCharArray( ), GetFullMatrix( ), GetWorkspaceData( ), PutCharArray( ), PutFullMatrix( )和 PutWorkspaceData 实现。

10.2.2 自动化服务应用举例一

【例 10-1】本例将通过 MATLAB 的自动化服务实现与 Delphi 的混合编程。源代码可以在配书光盘的 sourcecode\example10-1 目录下找到。

1. 编写 MATLAB 函数文件

首先在 MATLAB 环境下编写一个 M 文件，用来绘制某学生的成绩饼图。文件名为 ddeexample.m，其代码如下：

```
function ddeexample()% 学生成绩饼图绘制程序
x=[78 89 61 98];
explode=zeros(size(x));
[c,offset]=min(x); %求最小值的下标 offset.
explode(offset)=1; %指定占比例最小的一块和整个饼图脱离
h=pie(x,explode);
textObjs=findobj(h,'Type','text');
oldStr=get(textObjs,{'String'});
Names={'数学 78';'语文 89';'英语 61';'物理 98'};
set(textObjs,{'String'},Names);
```

该程序在 MATLAB 环境下的运行结果如图 10-3 所示。

2. 在 Delphi 中建立一个项目来实现 MATLAB 自动化服务功能

第一步：启动 Delphi 6.0，从“File”菜单中选择“New Application”子菜单，生成一个空的应用程序，在 Form1 的 Caption 属性处输入“通过自动化服务调用 MATLAB”，并加入两个按钮控件 Button1 和 Button2，然后修改按钮的 Caption 属性。Delphi 程序主窗体如图 10-4 所示。

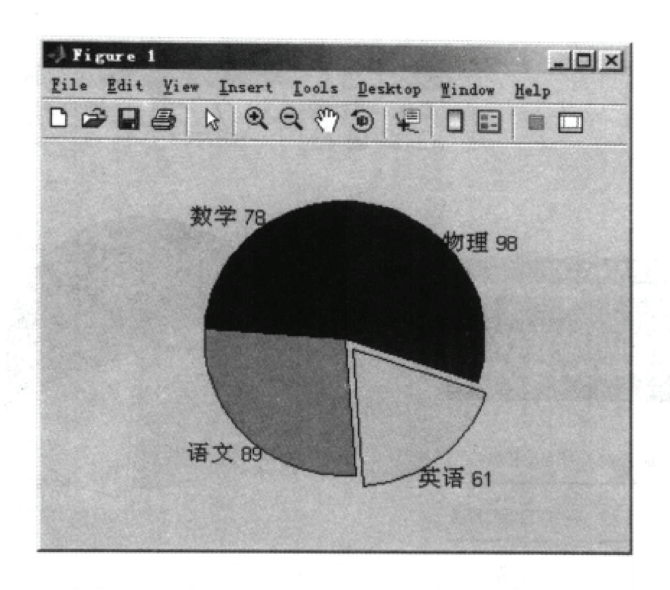


图 10-3　例 10-1 的 MATLAB 环境运行结果

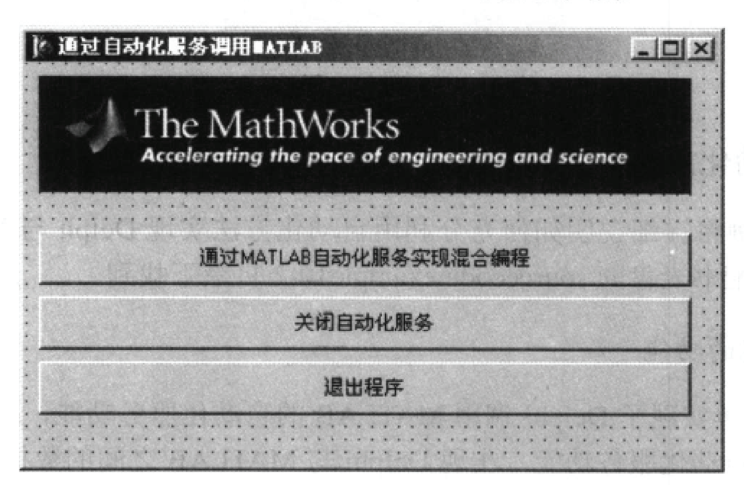


图 10-4　例 10-1 的 Delphi 程序主窗体

第二步：在 Uses 单元中加入 comobj 单元，在 Form1 上双击“Button1”按钮，编辑该按钮的 OnClick 过程，并添加相应的代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MATLAB: olevariant;
begin
  MATLAB:=CreateoleObject('MATLAB.Application');
  MATLAB.minimizecommandwindow;
  MATLAB.execute('ddeexample');
end;
```

第三步：编译并点击 Delphi 的▶按钮运行程序，单击“通过 MATLAB 自动化服务实现混合编程”按钮，运行结果如图 10-5 所示。

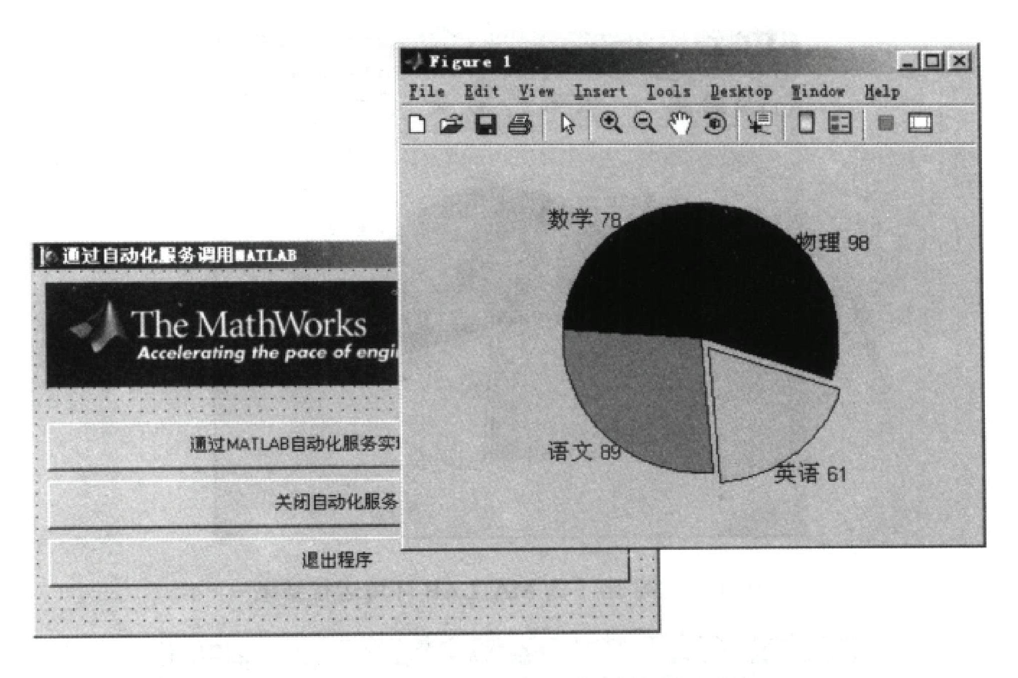


图 10-5　例 10-1 的 Delphi 主程序运行结果

10.2.3　自动化服务应用举例二

【例 10-2】本例将详细说明如何充分利用自动化方法实现 Delph 与 MATLAB 的交互。本例源代码可以在配书光盘的 sourcecode\example10-2 目录下找到。

1. 本例要实现的功能

例 10-1 只是简单实现了 Delphi 调用 MATLAB 的自动化服务功能，没有实现两者的数据交互，本例将通过自动化服务功能，实现 Delphi 与 MATLAB 之间的数据传送。

第一步：与例 10-1 类似，先在 Delphi 环境下建立一个新的项目，添加按钮等控件，然后添加按钮对应的 OnClick 事件，完成后的 Delphi 程序主窗体如图 10-6 所示。

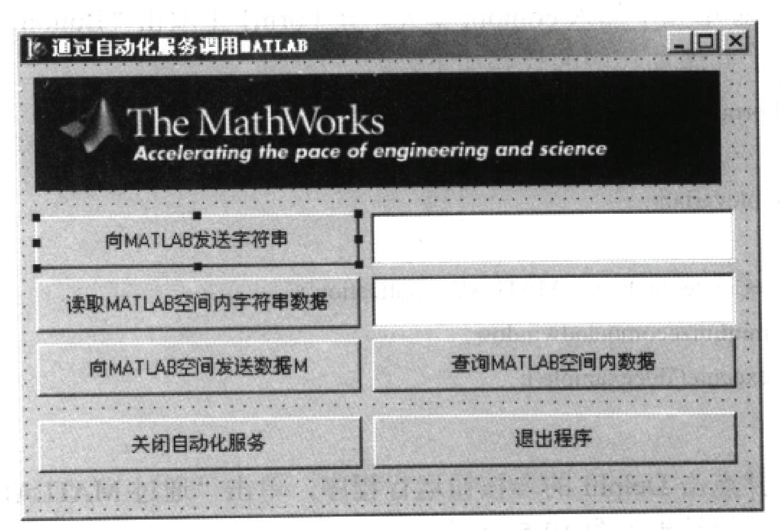


图 10-6　例 10-2 的 Delphi 程序主窗体

源代码如下：

```
unit Unit1;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
     Dialogs, StdCtrls, jpeg, ExtCtrls,comobj;
    type
     TForm1 = class(TForm)
     Button2: TButton;
     Image1: TImage;
     Button3: TButton;
     Button4: TButton;
     Edit1: TEdit;
     Button5: TButton;
     Edit2: TEdit;
     Button6: TButton;
     Button7: TButton;
     procedure Button2Click(Sender: TObject);
     procedure Button3Click(Sender: TObject);
     procedure Button4Click(Sender: TObject);
     procedure Button5Click(Sender: TObject);
     procedure Button6Click(Sender: TObject);
     procedure Button7Click(Sender: TObject);
     procedure FormCreate(Sender: TObject);
 private
     Matlab: olevariant;
 public
     { Public declarations }
   end;
 var
       Form1: TForm1;
 implementation
 {$R *.dfm}

 procedure TForm1.Button2Click(Sender: TObject);
 begin
   Matlab.quit;
 end;

 procedure TForm1.Button3Click(Sender: TObject);
 begin
   close;
```

```
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
        MATLAB.PutCharArray('str', 'base',edit1.Text);
end;

procedure TForm1.Button5Click(Sender: TObject);
var
        S:    String;
begin
        S:=MATLAB.GetCharArray('str', 'base');
        edit2.Text:=S;
end;

procedure TForm1.Button6Click(Sender: TObject);
var
   M: variant;
   i: integer;
begin
        M:=VarArrayCreate([0,100], varInteger);
        for i:=0 to 100 do
              begin
                      M[i]:=i*20;
         end;
        MATLAB.PutWorkspaceData('M', 'global', M);
end;

procedure TForm1.Button7Click(Sender: TObject);
var
        result : String;
begin
        result := MATLAB.Execute('whos global');
        showmessage(result);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
        MATLAB:=CreateoleObject('MATLAB.Application');
        MATLAB.minimizecommandwindow;
end;
end.
```

第二步：编译并运行程序，单击“通过 MATLAB 自动化服务实现混合编程”按钮，运行结果如图 10-7 所示。在编辑框输入字符串“测试应用程序”，单击“向 MATLAB 发送字符串”按钮，然后单击“读取 MATLAB 空间内字符串数据”，右侧编辑框正确显示刚才发送的字符串，表明已经成功实现了 Delphi 和 MATLAB 的通信。单击“向 MATLAB 空间发送数据 M”按钮，然后在 MATLAB 的提示符下输入“M”，即可以查询 M 变量，结果如图 10-8 所示。通过单击“查询 MATLAB 空间内数据”按钮，可以查询 MATLAB 空间内变量的属性，查询结果如图 10-9 所示。

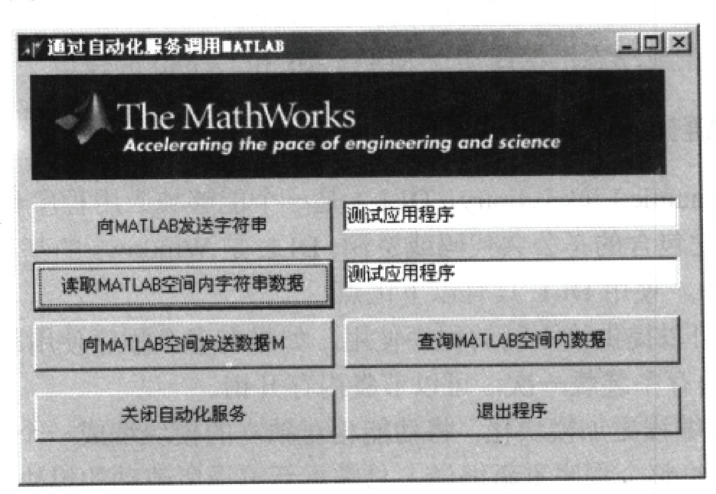


图 10-7　例 10-2 的程序运行结果

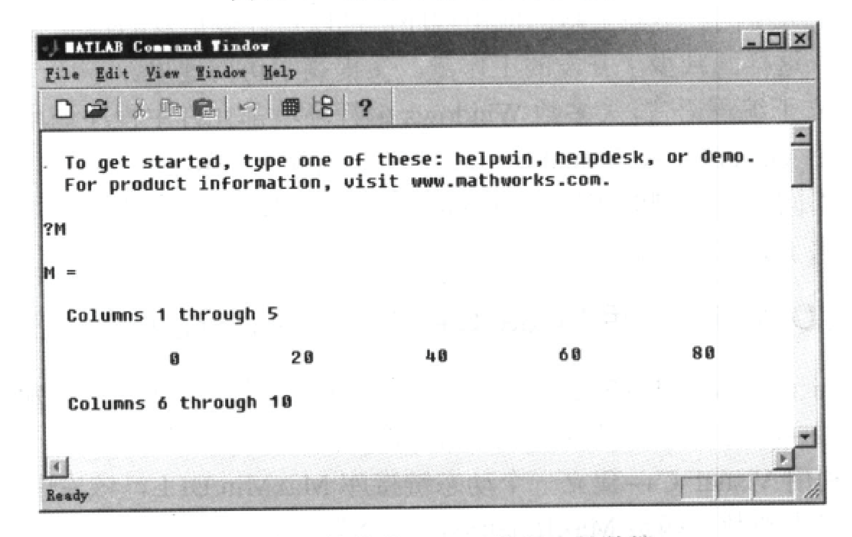


图 10-8　接收从 Delphi 发送来的数据

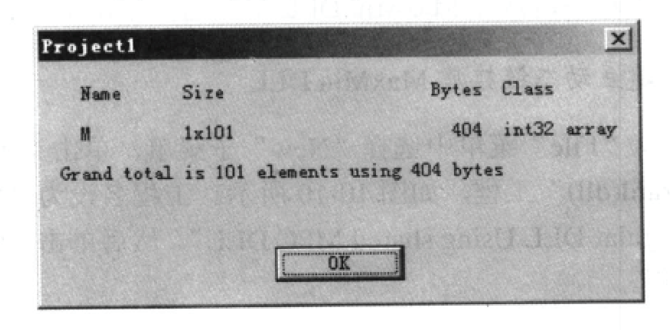


图 10-9　MATLAB 空间内变量属性的查询结果

10.3 利用 MATLAB 引擎实现混合编程

MATLAB 引擎本身并没有直接提供与 Delphi 的应用程序接口，但是提供了基于 Win32 平台的 C/C++应用程序接口，可以利用 MATLAB 与 C/C++的接口函数，通过 MATLAB 引擎进行指令处理和数据传递。为此，需要编写出 C++语言的动态链接库，作为 MATLAB 与 Delphi 的接口，然后在 Delphi 中进行 C++动态库函数的调用。MATLAB 引擎在第 3 章进行了详细的介绍，这里不再赘述。本章主要讲述如何利用 Visual C++开发包含 MATLAB 引擎函数的动态链接库，以及如何生成动态链接库实现 MATLAB 和 Delphi 的接口。

10.3.1 动态链接库介绍

动态链接库（Dynamic Link Library，DLL）是一个能够被应用程序和其他 DLL 调用的过程和函数的集合体，它包含的是公共代码或资源。DLL 是 Windows 的基石，所有的 Win32API 函数都包含在 DLL 中。使用 DLL 具有以下优点：

- 一个 DLL 可以提供给不同的程序使用，如果有多个程序使用相同的 DLL，也只需将 DLL 在内存中装载一次，可以节省内存开销。
- DLL 可以使编程更加模块化，将功能相对独立的模块编成一个动态链接库，改动程序时不需要将整个程序重新编译，只需重新编译所改动的模块。
- 使用了 DLL 组件包可以大大减小可执行文件的规模。
- 对于一个大型的、不断更新的应用程序，可以将许多重复的功能写成 DLL，用主程序调用，这样既减少了开发的工作量，又提高了访问速度。
- DLL 独立于编程语言，大多数 Windows 编程环境都允许主程序调用 DLL 中的函数，即可以用 Visual C++、Visual Basic、PowerBuilder、Delphi、汇编语言等建立 DLL，然后在不同语言编制的应用程序中调用它。因此，为多人使用不同的编程语言开发项目提供了极大的方便。

10.3.2 在 Delphi 中调用 Visual C++创建的动态链接库的实例

在讲述 Delphi 调用 MATLAB 引擎动态链接库之前，首先介绍一个实例，以帮助读者对 Delphi 和 Visual C++通过 DLL 混合编程有一个比较直观的了解。

【例 10-3】 利用 Visual C++建立一个动态链接库 MaxMin.DLL，该库中包含两个函数：返回三个整数中最大整数的函数 Max1( )和返回三个整数中最小整数的函数 Min1( )。然后用 Delphi 编写测试程序调用动态链接库 MaxMin.DLL 中的这两个函数。具体包含以下 5 个步骤。

1. Visual C++ 6.0 建立动态链接库 MaxMin.DLL

启动 Visual C++，从“File”菜单中选择“New”子菜单，单击“Project”标签。选择创建一个“MFC AppWizard(dll)”工程，如图 10-10 所示。工程名设为“Maxmin”，按“OK”按钮后，选择生成“Regular DLL Using shared MFC DLL”，然后单击“Finish”按钮，即创造了一个 DLL 的框架工程。

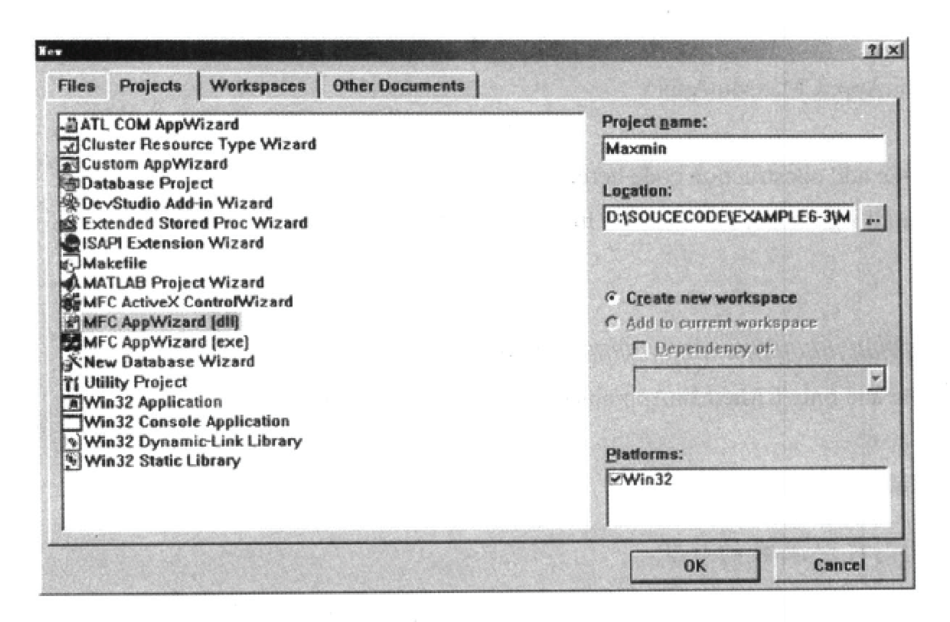


图 10-10　Visual C++向导菜单

2. 编辑 Maxmin.cpp 文件

AppWizard 产生下面的代码，其中包含了 CwinApp 的派生类。

```
// MaxMin.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "MaxMin.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMaxMinApp

BEGIN_MESSAGE_MAP(CMaxMinApp, CWinApp)
  //{{AFX_MSG_MAP(CMaxMinApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMaxMinApp construction
```

```
CMaxMinApp::CMaxMinApp()
{
  // TODO: add construction code here,
  // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CMaxMinApp object

CMaxMinApp theApp;
```

3. 加入代码实现函数的导出

把下面的代码加入到 Maxmin.cpp 文件中（也可以使用一个新的文件）：

```
extern "C" __declspec(dllexport) int Min1(int x,int y,int z)
{
if ((x<=y) & (x<=z)) return x;
else if ((y<=x) & (y<=z)) return y;
else return z;   /*找出 x，y，z 中的最小整数*/
}

extern "C" __declspec(dllexport) int Max1(int x,int y,int z)
{
if ((x>=y) & (x>=z)) return x;
else if ((y>=x) & (y>=z)) return y;
else return z;   /*找出 x，y，z 中的最大整数*/
}
```

4. 编译工程

编译该工程，生成 Maxmin.DLL 文件。

5. 用 Delphi 编写调用 Maxmin.DLL 的测试程序

编写 DLL 的目的是为了输出例程供其他程序调用，因此在 DLL 的工程文件中要把输出的例程用 Exports 关键字引出。在调用 DLL 的应用程序中，需要声明用到的 DLL 中的方法，声明格式要和 DLL 中的声明一样。访问 DLL 中的例程可采用静态调用和动态调用两种方式。静态调用方式就是在单元的 Interface 部分用 External 指示字列出要从 DLL 中引入的例程；动态调用方式则是通过调用 Windows 的 API，包括 LoadLibrary( )函数、GetProcAddress( )函数以及 FreeLibrary( )函数，动态地引入 DLL 中的例程。

（1）静态调用

第一步：启动 Delphi，选择“New Application”，生成一个空的应用程序，在 Form 的 Caption 属性处输入“Delphi 调用 Visual C++生成的 DLL”，在 Form 中加入控件，如图 10-11

所示（其中所有 Edit 控件的“Text”属性均设为空）。

图 10-11　Delphi 程序主窗体

第二步：选择“File”菜单中的“New…”子菜单，在出现的“New Item”对话框中选择“unit”，单击“OK”按钮，生成一个空的源文件，在该文件中输入以下内容：

```
unit maxmin;

interface
function Min1(x,y,z:Integer):Integer; cdecl;
function Max1(x,y,z:Integer):Integer; cdecl;

implementation
function Min1;external 'MaxMin.DLL' name 'Min1';
function Max1;external 'MaxMin.DLL' name 'Max1';
end.
```

第三步：在 Unit1.pas 文件中的“uses”中加入“Maxmin”。

第四步：在 Form 上双击“静态调用”按钮，对该按钮的“Click”事件编程，代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
editMax.Text:=IntToStr(Max1(StrToInt(edit1.Text),
StrToInt(edit2.Text),StrToInt(edit3.Text))); //调用动态链接库中的函数 Max1
editMin.Text:=IntToStr(Min1(StrToInt(edit1.Text),
StrToInt(edit2.Text),StrToInt(edit3.Text)));
end;
```

第五步：将上述步骤中 Visual C++所建立的动态链接库“Maxmin.DLL”拷贝到 Delphi 的当前工作目录中。然后运行 Delphi 创建的测试程序，运行结果如图 10-12 所示。

图 10-12　Delphi 程序静态调用运行结果

（2）动态调用

第一步：选择“File”菜单中的“New…”子菜单，在出现的“New Item”对话框中选择“unit”，单击“OK”按钮，生成一个空的源文件，在该文件中输入以下内容：

```
unit maxmin2;
 unit maxmin2;


interface


type
TMin1=function(x,y,z:Integer):Integer; cdecl;
TMax1=function(x,y,z:Integer):Integer; cdecl;
THandle=Integer;


implementation
 end.
```

选择“File”菜单的“Save As…”子菜单，将上述文件存为“Maxmin2.pas”。

第二步：在 Unit1.pas 文件中的“uses”中加入“Maxmin2”。

第三步：在 Form 上双击“动态调用”按钮，为该按钮添加“Click”事件代码如下：

```
procedure TForm1.Button2Click(Sender: TObject);
var
Handle:THandle;
Min1:TMin1;
Max1:TMax1;
begin
Handle:=LoadLibrary('MaxMin.dll'); //将“MaxMin.dll”的文件映像映射进调用进程的地址空间
if Handle<>0 then
begin
@Min1:=GetProcAddress(Handle,'Min1'); //取得 DLL 中函数 Min1( )的地址
```

```
@Max1:=GetProcAddress(Handle,'Max1'); //取得 DLL 中函数 Max1( )的地址
if (@Min1<>nil) and (@Min1<>nil) then
begin
editMin.Text:=IntToStr(Min1(StrToInt(edit1.Text),
StrToInt(edit2.Text),StrToInt(edit3.Text))); //调用动态链接库中的函数 Min1
editMax.Text:=IntToStr(Max1(StrToInt(edit1.Text),
StrToInt(edit2.Text),StrToInt(edit3.Text))); //调用动态链接库中的函数 Max1
end else ShowMessage('调用函数 "GetProcAddress" 时出错！');
FreeLibrary(Handle); //从进程的地址空间中解除 "MaxMin.dll" 文件的映射
end;
end;
```

第四步：运行 Delphi 创建的测试程序，单击“动态调用”按钮，运行结果如图 10-13 所示。

图 10-13 Delphi 程序动态调用运行结果

10.3.3 MATLAB 引擎动态链接库的设计

MATLAB 引擎本身并没有直接提供与 Delphi 的应用程序接口，但是提供了基于 Win32 平台的 C/C++应用程序接口。为此，需要首先利用 MATLAB 与 C/C++的接口函数，编写 C++ 语言的动态链接库。在动态链接库中，需要将 Delphi 中的数据送入 MATLAB 进行分析，并选择合适的处理函数进行运算，然后将结果送回 Delphi。为了实现上述目的，我们在动态链接库中设计了几个函数，读者也可以根据需要添加相应的处理函数。

【例 10-4】利用 Visual C++建立 MATAB 引擎库的动态链接库，具体包含以下几个步骤：

（1）利用 Visual C++ Developer Studio 创建 DLL。

启动 Visual C++，从 File 菜单中选择“New”，单击 Project 标签。选择 MFC AppWizard (dll)，工程名设为“MATLABengine”，按“OK”按钮后，选择“Regular DLL Using shared MFC DLL”，其他按默认值设置。

（2）编辑 MATLABengine.cpp 文件，加入以下代码到 MaxMin.cpp 文件中，实现函数的导出。

```
//头文件
#include "engine.h"
```

```
Engine *ep=NULL;
mxArray *T = NULL;
//打开 MATLAB 引擎的函数
extern "C" __declspec(dllexport) int MatOpenEng(char *StartCmd)
{
 if(ep) return 0;//如果已打开则退出
 if(!(ep=engOpen(StartCmd))) return -1;
 else return 1;
 return 1;
}
//关闭 MATLAB 引擎的函数
extern "C" __declspec(dllexport) int MatCloseEng(void)
{
  int Result=engClose(ep);
  if (Result==0)    //Success
        ep=NULL;
  return Result;
}
//执行 MATLAB 命令函数
extern "C" __declspec(dllexport) int MatEvalString(char *string)
{
  return (engEvalString(ep, string));
}

extern "C" __declspec(dllexport) int MatGetVisible(bool* value)
{
  return (engGetVisible(ep, value));
}

extern "C" __declspec(dllexport) int MatSetVisible(bool value)
{
  return (engSetVisible(ep, value));
}
//发送数据到 MATLAB 的函数
extern "C" __declspec(dllexport) int MatPutVariable(char *cc,int num,double dd[])
{
  if(!ep) return 0;
  if(num<1) return -1;
  T=mxCreateDoubleMatrix(1,num,mxREAL);
  memcpy((char *)mxGetPr(T),(char *)dd,num * sizeof(double));
```

```
  engPutVariable(ep,cc,T);
  return 1;
}
//获取 MATLAB 数据函数
extern "C" __declspec(dllexport) int MatGetVariable(char *cc,int num,double dd[])
{
  mxArray *result;
  if(!ep)return 0;
  if(num<1)return-1;
  result=engGetVariable(ep,cc);
  memcpy((char *)dd,(char *)mxGetPr(result),num * sizeof(double));
  return 0;
}
```

读者还可以根据需要定义其他一些函数，例如获取变量名函数等。

（3）动态链接库在 Delphi 中的应用。

① 为了在 Delphi 中使用动态链接库函数，首先要对这些函数进行如下声明：

```
unit MATLABengine;

interface
function  MatOpenEng(p:PChar):Integer;cdecl;
function  MatCloseEng():Integer;cdecl;
function  MatEvalString(p:PChar):Integer;cdecl;
function  MatGetVisible(value:Boolean):Integer;cdecl;
function  MatSetVisible(value:Boolean):Integer;cdecl;
function  MatPutVariable(cc:Pchar;num:integer;dd: array of double):Integer;cdecl;
function  MatGetVariable(cc:Pchar;num:integer;dd: array of double):Integer;cdecl;

implementation
function  MatOpenEng; external 'matenginedll.dll';
function  MatCloseEng;external 'matenginedll.dll';
function  MatEvalString;external 'matenginedll.dll';
function  MatGetVisible;external 'matenginedll.dll';
function  MatSetVisible;external 'matenginedll.dll';
function  MatPutVariable;external 'matenginedll.dll';
function  MatGetVariable;external 'matenginedll.dll';
end.
```

② 启动 Delphi，选择“New Application”，生成一个空的应用程序，在 Form 的“Caption”属性处输入“Delphi 通过 DLL 调用 MATLAB 引擎”，在 Form 中添加控件，如图 10-14 所示。

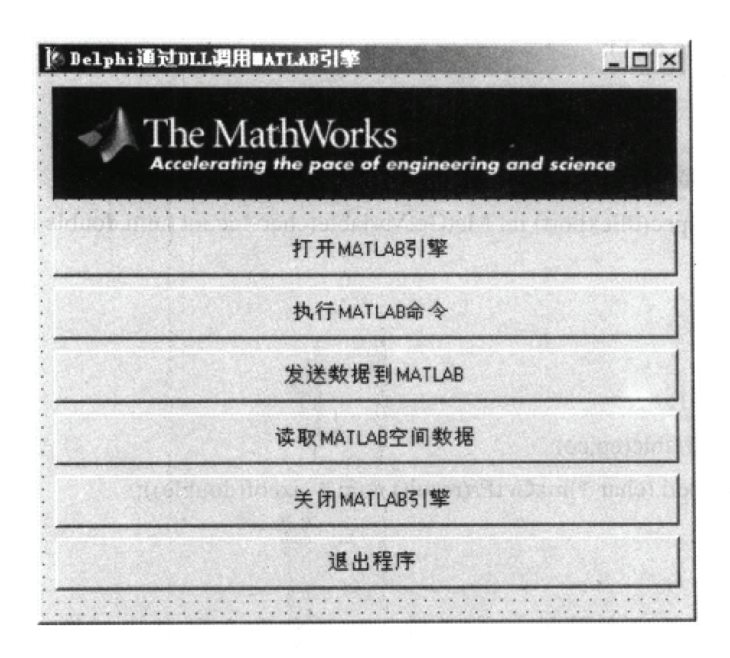


图 10-14　Delphi 程序主窗体

③ 添加代码，实现功能调用：

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,MATLABengine, jpeg, ExtCtrls;

type
    TForm1 = class(TForm)
    Button1: TButton;
    Image1: TImage;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  result : integer;

begin
  result:=MatOpenEng(PChar('\0'));

end;

procedure TForm1.Button2Click(Sender: TObject);
begin
          MatCloseEng();
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
          close;
end;

procedure TForm1.Button4Click(Sender: TObject);
var
  xx:   array[0..9]of double;
begin
  MatGetVariable('test',10,xx);
  showmessage(currtostr(xx[5]));//显示数组中第六个元素的值
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
   MatEvalString(PChar('mesh(peaks)'));
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);
var
  dd:     array[0..9]of double;
  i:      integer;
begin
  for i:=0 to 9 do
  begin
  dd[i]:=i;
  end;
  MatPutVariable('test',10,dd);
end;

end.
```

④ 将上述步骤中 Visual C++所建立的动态链接库 MATLABengine.dll 拷贝到当前工作目录中，然后运行 Delphi 创建的测试程序，单击各按钮即可查看运行结果。

10.4 Delphi 调用 Mideva 生成的动态链接库

10.4.1 Mideva 介绍

Mideva 是 MathTools 推出的一种 MATLAB 编译开发软件平台，提供对 MATLAB 程序文件（M 文件）的解释执行和开发环境支持，它集编辑、调试、编译和优化于一体。该软件有为 Borland C++、Visual C++等编程语言开发的不同版本。Mideva 提供的功能相当强大，因为它包含了近千个 MATLAB 的基本功能函数，通过必要的设置就可以直接实现与 C++的混合编程，而不必再依赖 MATLAB；同时，Mideva 还提供编译转换功能，能够将 MATLAB 函数或编写的 MATLAB 程序转换为 EXE 或者 DLL，从而可以脱离 MATLAB 环境实现对 MATLAB 函数的调用。Mideva 的详细介绍见第 5 章。

虽然 Mideva 提供了输出 DLL 文件的功能，但是在实际应用中输出的 DLL 文件往往不能直接使用。因此，本节通过实例介绍利用 Mideva 输出 DLL 文件过程中伴随生成的 CPP 文件，手动编写动态链接库，然后在 Delphi 中调用，从而实现 Mideva 和 Delphi 的混合编程。

10.4.2 应用实例

【例 10-5】本例将实现 Delphi 和 Mideva 的混合编程。虽然本节也是利用动态链接库的方式实现混合编程，但是它不同于 10.3 节的方法。本节直接利用 Mideva 生成的 CPP 文件来创建 DLL。具体包含以下步骤：

（1）首先准备好 M 文件，用来实现点沿圆周运动的动画。

在 Mideva 环境下编写 diamond.m 文件，其源代码如下：

```
function diamond()
%实现点沿圆周运动的动画
t=0:pi/2000:pi*2;
```

```
x=sin(t);
y=cos(t);
plot(y,x,'b');
axis equal
n=length(t);
h=line('color','g','linestyle','.',...
    'marker','.','markersize',50,'erasemode','xor');
i=1;
while i
    set(h,'xdata',y(i),'ydata',x(i));
    drawnow;
    i=i+1;
    if i>n
        i=1;
    end
end
```

该程序在 Mideva 环境下的运行结果如图 10-15 所示。

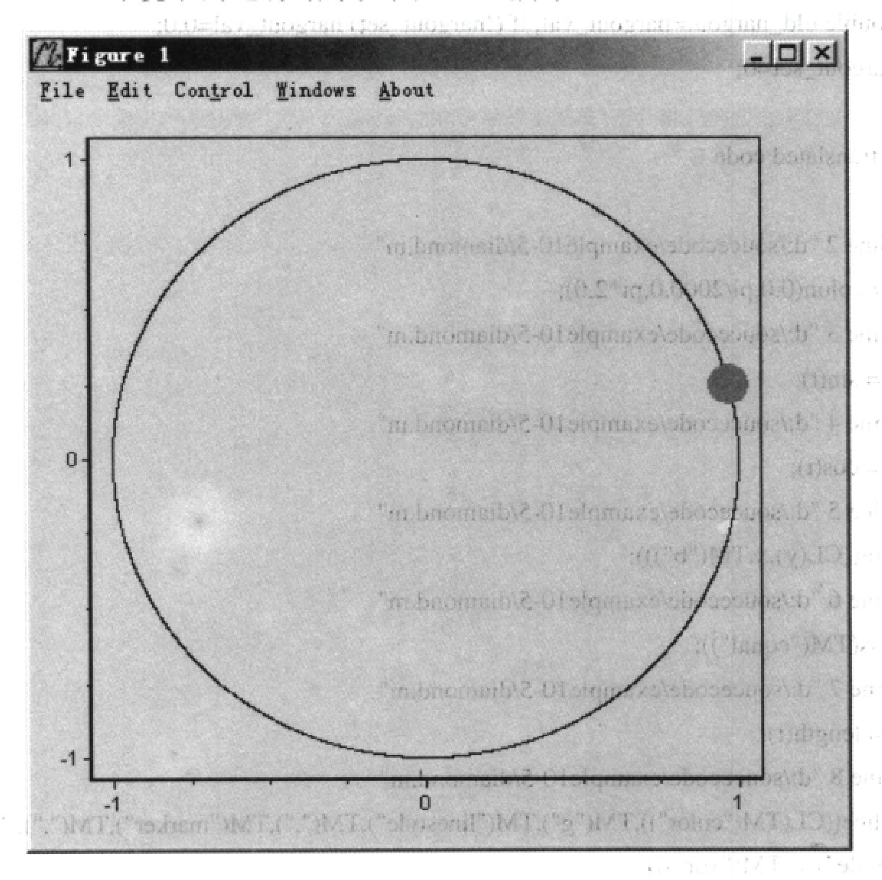


图 10-15　diamond.m 程序在 Mideva 环境下的运行结果

（2）用 Mideva 将 diamond.m 转换为 DLL。

单击 Mideva 环境“File”菜单下“Compile to dll”子菜单，在弹出的“go command”对话框中选择要编译的 M 文件，即 diamond.m，编译完毕后，将生成对应的 DLL 文件 diamond.dll

及一系列 CPP 文件，其中 diamond.cpp 文件的内容如下：

```
#include "matlib.h"
#pragma hdrstop

#include "diamond.h"


Mm diamond() {
   begin_scope
   dMm(t); dMm(x); dMm(y); dMm(n); dMm(h); dMm(i_);

   #line 1 "d:/soucecode/example10-5/diamond.m"
   call_stack_begin;
   #line 1 "d:/soucecode/example10-5/diamond.m"
   // nargin, nargout entry code
   double old_nargin=nargin_val; if (!nargin_set) nargin_val=0.0;
   nargin_set=0;
   double old_nargout=nargout_val; if (!nargout_set) nargout_val=0.0;
   nargout_set=0;

   // translated code

   #line 2 "d:/soucecode/example10-5/diamond.m"
_  t = colon(0.0,pi/2000.0,pi*2.0);
   #line 3 "d:/soucecode/example10-5/diamond.m"
_  x = sin(t);
   #line 4 "d:/soucecode/example10-5/diamond.m"
_  y = cos(t);
   #line 5 "d:/soucecode/example10-5/diamond.m"
_  plot((CL(y),x,TM("b")));
   #line 6 "d:/soucecode/example10-5/diamond.m"
_  axis(TM("equal"));
   #line 7 "d:/soucecode/example10-5/diamond.m"
_  n = length(t);
   #line 8 "d:/soucecode/example10-5/diamond.m"
_  h=line((CL(TM("color")),TM("g"),TM("linestyle"),TM("."),TM("marker"),TM("."), TM("markersize"),
50.0,TM("erasemode")    ,TM("xor")));
   #line 10 "d:/soucecode/example10-5/diamond.m"
_  i_ = 1.0;
   #line 11 "d:/soucecode/example10-5/diamond.m"
_  while (istrue(i_)) {
      #line 12 "d:/soucecode/example10-5/diamond.m"
```

```
_       set(h,(CL(TM("xdata")),y(i_),TM("ydata"),x(i_)));
        #line 13 "d:/soucecode/example10-5/diamond.m"
_       drawnow();
        #line 14 "d:/soucecode/example10-5/diamond.m"
_       i_ = i_+1.0;
        #line 15 "d:/soucecode/example10-5/diamond.m"
_       if (istrue(i_>n)) {
          #line 16 "d:/soucecode/example10-5/diamond.m"
_         i_ = 1.0;
          #line 17 "d:/soucecode/example10-5/diamond.m"
        }
        #line 18 "d:/soucecode/example10-5/diamond.m"
      }

      call_stack_end;

      // nargin, nargout exit code
      nargin_val=old_nargin; nargout_val=old_nargout;

      // function exit code

      return x_M;
      end_scope
    }
```

（3）利用 Visual C++ 6.0 生成动态链接库，包含以下 3 个步骤：

① 利用 MFC AppWizard（dll）创建一个项目，名字为“delphidll”，采用默认设置。

② 在头文件中输入以下内容：

```
extern "C" _declspec(dllexport) void diamond();
```

利用 Mideva 生成的 diamond.cpp 文件，在 delphidll.cpp 源程序文件中添加如下代码：

```
extern "C" _declspec(dllexport) void diamond()
{
  // TODO: add construction code here,
  // Place all significant initialization in InitInstance
   initM(Matcom_VERSION);
     begin_scope
     dMm(t); dMm(x); dMm(y); dMm(n); dMm(h); dMm(i_);

     #line 1 "d:/soucecode/example10-5/diamond.m"
     call_stack_begin;
     #line 1 "d:/soucecode/example10-5/diamond.m"
     // nargin, nargout entry code
```

```
double old_nargin=nargin_val; if (!nargin_set) nargin_val=0.0;
nargin_set=0;
double old_nargout=nargout_val; if (!nargout_set) nargout_val=0.0;
nargout_set=0;

// translated code

#line 2 "d:/soucecode/example10-5/diamond.m"
t = colon(0.0,pi/2000.0,pi*2.0);
#line 3 "d:/soucecode/example10-5/diamond.m"
x = sin(t);
#line 4 "d:/soucecode/example10-5/diamond.m"
y = cos(t);
#line 5 "d:/soucecode/example10-5/diamond.m"
plot((CL(y),x,TM("b")));
#line 6 "d:/soucecode/example10-5/diamond.m"
axis(TM("equal"));
#line 7 "d:/soucecode/example10-5/diamond.m"
n = length(t);
#line 8 "d:/soucecode/example10-5/diamond.m"
h = line((CL(TM("color")),TM("g"),TM("linestyle"),TM("."),TM("marker"),TM("."),
    TM("markersize"),50.0,TM("erasemode"),TM("xor")));
#line 10 "d:/soucecode/example10-5/diamond.m"
i_ = 1.0;
#line 11 "d:/soucecode/example10-5/diamond.m"
while (istrue(i_)) {
   #line 12 "d:/soucecode/example10-5/diamond.m"
   set(h,(CL(TM("xdata")),y(i_),TM("ydata"),x(i_)));
   #line 13 "d:/soucecode/example10-5/diamond.m"
   drawnow();
   #line 14 "d:/soucecode/example10-5/diamond.m"
   i_ = i_+1.0;
   #line 15 "d:/soucecode/example10-5/diamond.m"
   if (istrue(i_>n)) {
      #line 16 "d:/soucecode/example10-5/diamond.m"
      i_ = 1.0;
      #line 17 "d:/soucecode/example10-5/diamond.m"
   }
   #line 18 "d:/soucecode/example10-5/diamond.m"
}

call_stack_end;
```

```
    // nargin, nargout exit code
    nargin_val=old_nargin; nargout_val=old_nargout;

    // function exit code

  //  return x_M;
    end_scope
  exitM();
 }
```

③ 编译，生成 delphidll.dll 文件。

（4）用 Delphi 编写调用 delphidll.dll 的程序，包含以下 4 个步骤。

① 启动 Delphi 6.0，选择"New Application"，生成一个空的应用程序，在 Form1 的 Caption 属性处输入"Delphi 调用 MIDEVA 的动态链接库"，加入两个按钮控件 Button1 和 Button2，如图 10-16 所示。

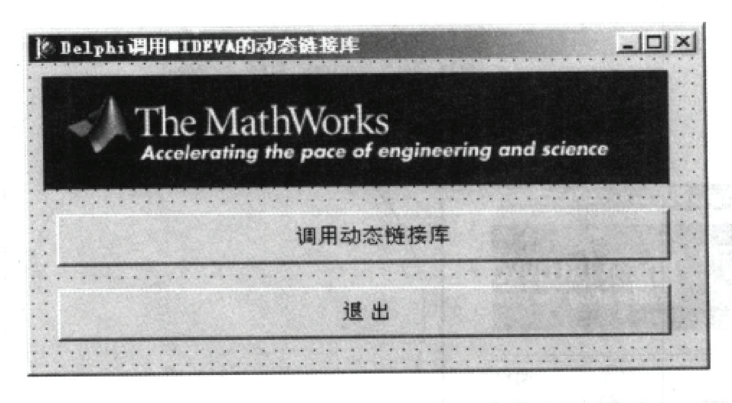


图 10-16　Delphi 程序主窗体

② 在源文件 unit1.pas 中输入以下内容：

```
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;
  Procedure diamond();
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);

  private
    { Private declarations }
```

```
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;
    Procedure diamond; external 'delphidll.dll' name 'diamond';
```

③ 在 Form1 上双击"调用动态链接库"按钮，添加该按钮的 Click 事件代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   diamond();
end;
```

④ 将 delphidll.dll 拷贝到 Delphi 的当前工作目录中。编译运行程序，运行结果如图 10-17 所示。

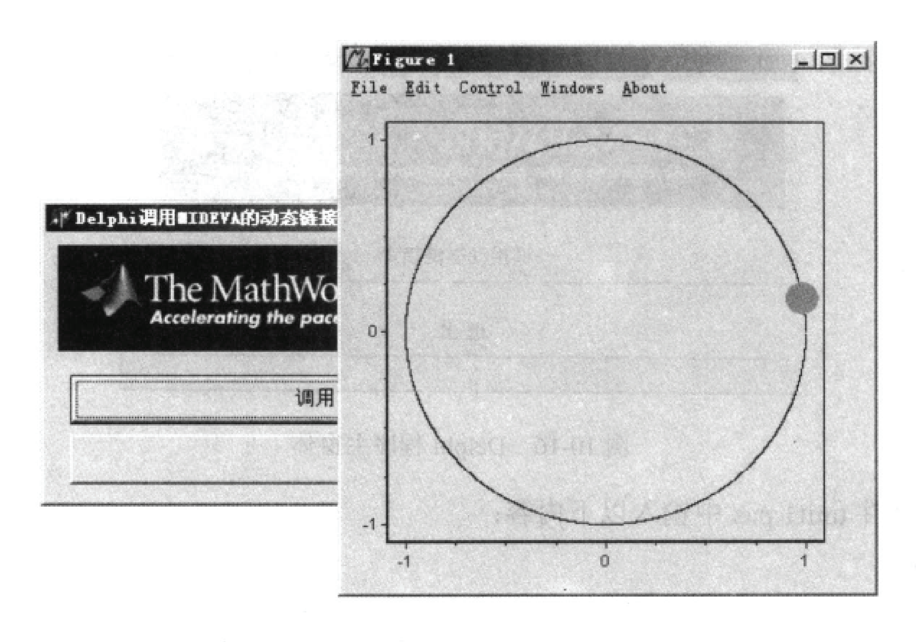


图 10-17　例 10-5 的 Delphi 程序运行结果

10.5　通过外部调用实现混合编程

10.5.1　外部调用方法介绍

外部调用方法就是可以通过三个 Windows SDK 函数 WinExec( )、ShellExecute( )和 CreateProcess( )来实现在 Delphi 的程序中加载其他执行程序。

WinExec( )函数原型如下：

```
UINT WinExec
(
    LPCSTR lpCmdLine,     // 命令行地址
```

```
    UINT uCmdShow // 窗口显示方式
);
```

ShellExecute( )函数原型如下：

```
HINSTANCE ShellExecute
(
    HWND hwnd, //父窗口句柄
      LPCTSTR lpOperation, // 操作类型指针
      LPCTSTR lpFile, // 文件/目录名指针
      LPCTSTR lpParameters, // 执行程序参数指针
      LPCTSTR lpDirectory, // 缺省目录指针
      INT nShowCmd     // 文件是否显示
);
```

CreateProcess( )函数原型如下：

```
BOOL CreateProcess
(
    LPCTSTR lpApplicationName,  // pointer to name of executable module
    LPTSTR lpCommandLine, // pointer to command line string
    LPSECURITY_ATTRIBUTES lpProcessAttributes,    // pointer to process security attributes
    LPSECURITY_ATTRIBUTES lpThreadAttributes,// pointer to thread security attributes
    BOOL bInheritHandles,    // handle inheritance flag
    DWORD dwCreationFlags, // creation flags
    LPVOID lpEnvironment,   // pointer to new environment block
    LPCTSTR lpCurrentDirectory,    // pointer to current directory name
    LPSTARTUPINFO lpStartupInfo,// pointer to STARTUPINFO
    LPPROCESS_INFORMATION lpProcessInformation //pointerto PROCESS_INFORMATION
);
```

函数 WinExec( )最简单，只有两个参数，前一个指定路径，后一个指定显示方式。其中，显示方式的参数中如果使用 SW_SHOWMAXMIZED 方式，则加载一个无最大化按钮的程序，不会出现正常的窗体，但实际上已经被加到任务列表里了。

ShellExecute( )较 WinExec( )稍微灵活一点，可以指定工作目录。下面的例子就是直接打开 c:\temp\1.txt，而不用加载与 txt 文件关联的应用程序。

```
shellExecute(NULL,NULL,'1.txt'),NULL,'c:\\temp',SW_SHOWMAXMIZED)
```

CreateProcess( )最复杂，有 10 个参数，不过大部分都可以用 NULL 代替，它可以指定进程的安全属性、继承信息、类的优先级等。

10.5.2 应用实例

【例 10-6】本例将实现 Delphi 外部调用执行程序，该执行程序由 M 函数转化而来。具体包含以下步骤：

（1）利用 Mideva 将 diamond.m 转换为 exe 文件，在“example10-5\Debug”目录下将会看到 diamond.exe 文件。

（2）启动 Delphi 6.0，选择“New Application”，生成一个空的应用程序，在 Form1 的 Caption 属性处输入“Delphi 外部调用”，并添加两个按钮控件 Button1 和 Button2，如图 10-18 所示。

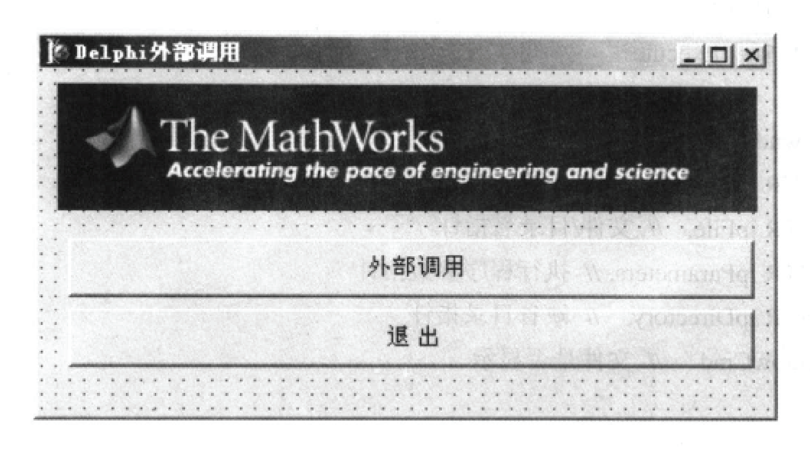


图 10-18 例 10-6 的 Delphi 程序主窗体

（3）在 Form1 上双击“外部调用”按钮，添加该按钮的 Click 事件代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  WinExec('diamond.exe',SW_SHOW); //外部调用
end;
```

（4）将 diamond.exe 拷贝到 Delphi 的当前工作目录中，编译运行程序就可以看到外部调用的效果了。

10.6 小　结

因为 Delphi 是很多开发人员喜欢使用的语言，本章对 Delphi 和 MATLAB 的混合编程方法进行了较详细的介绍，包含实现 Delphi 和 MATLAB 混合编程的三种具体方法，即通过 MATLAB 自动化服务、利用 MATLAB 引擎和 Delphi 调用 Mideva 生成的动态链接库。实际上，凡是 Visual C++与 MATLAB 可以实现的混合编程方法，几乎在 Delphi 中都可以实现。

第 11 章　MATLAB 和 Excel 的混合编程

11.1　引　　言

MATLAB 具有强大的数据计算和图形显示能力；Excel 则是 Microsoft 提供的一种常见的表格处理软件，它也具有强大的数据统计和显示能力。对于一些常见的统计图形显示，例如棒图、饼图、拆线图等，Excel 的显示质量很高。但是，对于一些较复杂的图形显示，例如等高线，Excel 则显得无能为力。在实际的工程应用中，通常有一些 Excel 表单（sheet）的数据，需要进行一些复杂的数据计算和处理后再进行统计或显示。另外，MATLAB 运算得到的结果，尽管可以利用 MATLAB 提供的函数进行显示，但其显示效果可能不如 Excel 易于控制，特别是在学术论文的写作过程中。此时，需要实现 MATLAB 和 Excel 的混合编程。

MATLAB 和 Excel 的混合编程有两种实现方式：一种是利用 MATLAB 6.5 以后版本提供的 Excel Link 插件，实现 Excel 和 MATLAB 的数据共享；另外一种方式是利用 MATLAB 提供的 Excel 生成器，可以生成 DLL 组件和 VBA 代码。其中，利用 DLL 组件可以进行 COM 生成器组件相似的操作；利用 VBA 代码则可以在 Excel 的 Visual Basic 编译器中直接使用，可以保存为插件。

11.2　通过 Excel Link 实现 Excel 和 MATLAB 的数据共享

11.2.1　概述

Excel Link 是一个软件插件，它可将 Excel 和 MATLAB 进行集成。通过对 Excel 和 MATLAB 的链接，用户可以在 Excel 的工作空间里，利用 Excel 的宏编程功能，使用 MATLAB 的数据处理和图形处理功能进行相关操作，同时 Excel 保证 MATLAB 和 Excel 工作空间中数据的交换和同步更新。使用 Excel Link 时，不必脱离 Excel 环境，而直接在 Excel 的工作区或者宏操作中调用 MATLAB 的函数。Excel 共提供了 11 个函数来实现链接和数据操作。

因此，通过 Excel Link 这个中介，Excel 成为 MATLAB 的一个易于使用的数据存储和应用开发前端，它是一个功能强大的计算和图形处理器。

Excel Link 的运行机制如图 11-1 所示。

11.2.2　Excel Link 的安装

Excel Link 对运行环境没有特别要求，只要能够同时运行 Microsoft Excel 和 MATLAB 即可。Excel Link 需要近 200KB 的硬盘空间，操作系统可以是 Windows 98, Windows XP 或 Windows 2000，此外还需要在 Windows 环境下先安装 Excel，然后再安装 MATLAB 和 Excel Link。

Excel Link 可以视为 MATLAB 的一个工具箱，Excel Link 的安装可按 MATLAB 安装向

导的提示进行，即在 MATLAB 安装组件选择框中一定要选择安装 Excel Link，并按提示继续安装。

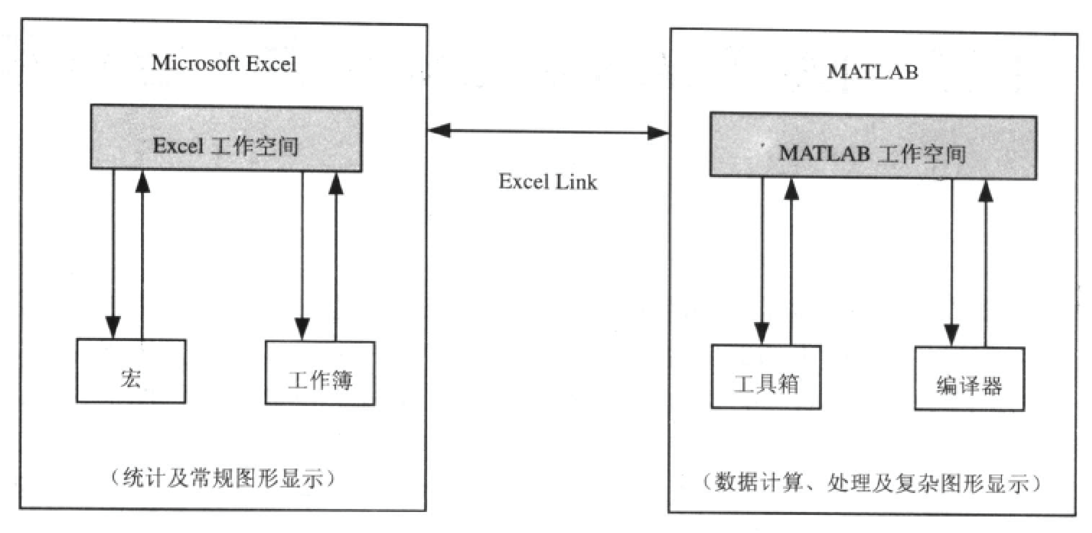


图 11-1　Excel Link 的运行机制

1. Excel 的设置

安装 Excel、MATLAB 及其工具箱 Excel Link 后，还需要在 Excel 中进行相应的设置，以便完成 Excel 和 MATLAB 的链接。具体需要完成以下步骤：

（1）启动 Excel。

（2）单击 Excel 的"工具"菜单。

（3）执行"加载宏"，如图 11-2 所示。

（4）在弹出的如图 11-3 所示的"加载宏"对话框中，单击"浏览"按钮。

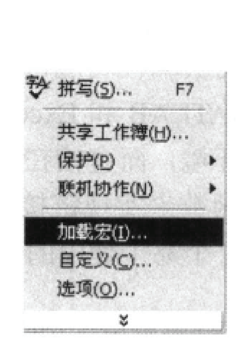


图 11-2　执行"加载宏"

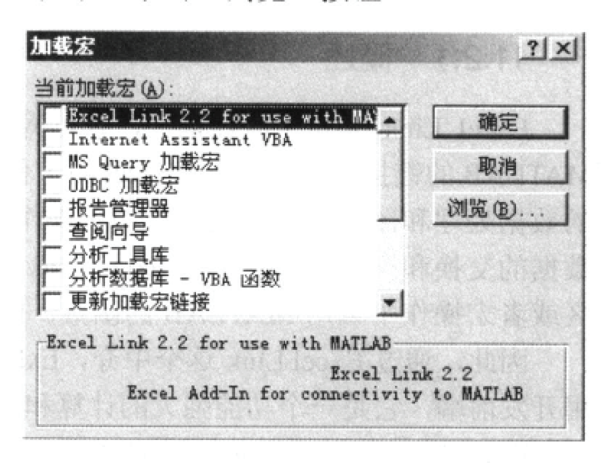


图 11-3　"加载宏"对话框

（5）在弹出的路径选择对话框中，选择 MATLAB 安装目录 toolbox\exlink 路径下的 excllink.xla，然后单击"确定"。

（6）在返回的"加载宏"对话框中，此时已经选中了 Excel Link。单击"确定"后 Excel Link 将加载 MATLAB。此时，可以看到 MATLAB 会自动启动，如图 11-4 所示。

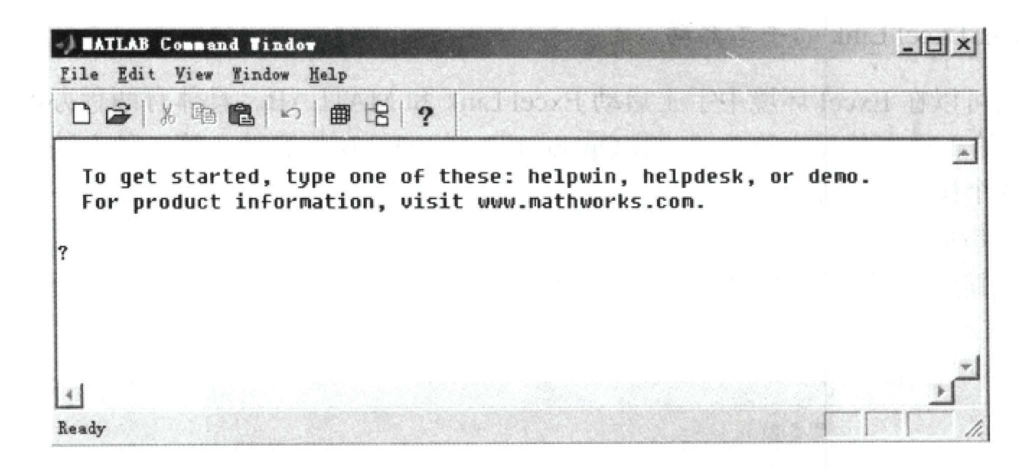


图 11-4 “MATLAB Command Window”窗口

稍后，Excel Link 工具栏会在 Excel 工作表中出现，如图 11-5 所示。添加的 Excel 工具栏中会出现 4 个执行 MATLAB 的命令按钮: startMATLAB、putmatrix、getmatrix 和 evalstring。它们分别表示启动 MATLAB、把数据传给 MATLAB、从 MATLAB 提取数据和执行 MATLAB 命令。

Excel 工具栏的 Excel Link 在不需要时，可以像其他的工具栏（例如“常用”或“格式”）一样取消，方法是在工具栏上单击鼠标右键，在弹出的对话框中去掉“Excel Link”。

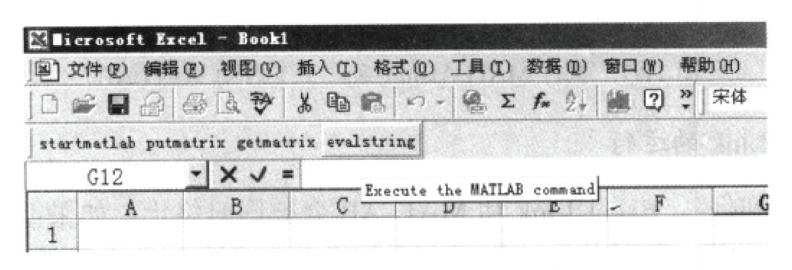


图 11-5 Excel Link 工具条

2. 设置 Excel Link 的自启动

在安装并设置 Excel Link 后，每次启动 Excel 时 Excel Link 和 MATLAB 都将自动运行。但有时使用者可能并不希望启动 Excel 时 MATLAB 也自动启动，那么可以通过在 Excel 数据表单元中输入“MLAutoStart("no")”实现关闭 MATLAB 的自动启动，如图 11-6 所示。该函数将会改变 Excel 初始化文件中对自动启动 Excel Link 和 MATLAB 的设置。当再次启动该文件时，Excel Link 和 MATLAB 不会自动运行。

图 11-6 关闭启动 Excel 时自动启动 MATLAB

3. 设置 Excel Link 的手工启动

用户也可以在 Excel 环境中手工启动 Excel Link 和 MATLAB。具体有两种办法：第一种是在 Excel 的工作表单元内输入“=MLOpen( )”，MATLAB 将启动。第二种方法稍微复杂一些，它由两个操作步骤组成：

（1）单击 Excel 的“工具”菜单，选择“宏”菜单项；

（2）在弹出的“宏”对话框中输入“MATLABinit”，如图 11-7 所示，然后单击“执行”按钮即可。

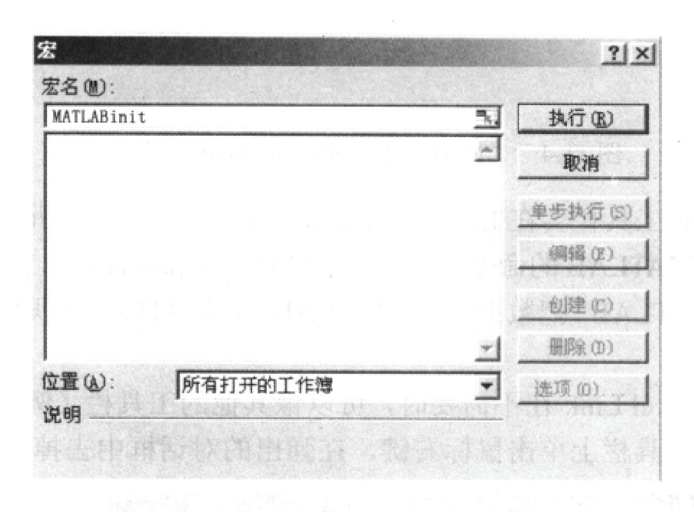


图 11-7　执行 MATLABinit 宏，手工启动 MATLAB

4. 终止 Excel Link 的运行

当终止 Excel 的时候，Excel Link 和 MATLAB 会被同时终止。如果需要在 Excel 环境中终止 Excel Link 和 MATLAB 的运行，则可在工作表单元内输入“=MLClose( )”。注意：“=”必须输入。

当需要再次重新启动 Excel Link 和 MATLAB 时，可以选择用前文提到的“＝MLOpen( )”或“MATLABinit”进行。

如果用户直接关闭了 MATLAB 的主窗口，而 Excel 继续运行，就可以直接向 Excel =MLClose( )。MLClose( )命令将通知 Excel MATLAB 已经停止运行。

11.2.3　Excel Link 的函数

利用 Excel Link, Excel 就成了 MATLAB 的一个功能强大的数据存储和应用终端。Excel Link 提供了一系列的连接管理和数据处理函数来支持 Excel 和 MATLAB 的连接，使得用户不必脱离 Excel 环境，而只需要以数据表单元的函数形式或宏命令来使用 MATLAB 提供的相关处理函数。

1. 连接管理函数

Excel Link 共提供了 4 个连接管理函数，实现对 Excel Link 的初始化、启动和终止，见表 11-1。其中，MATLABinit 只能以宏命令的方式运行，其他命令均可以作为数据单元函数或者宏命令来执行。

MLAutoStart 用来设置自启动，系统对 Excel Link 和 MATLAB 启动方式的默认设置是在

每一次启动 Excel 时自动启动，如果选择手动启动，就需要 MATLABinit 来初始化 Excel Link 并且启动 MATLAB。MLClose 在保持 Excel 继续运行的状态下，用来终止 MATLAB 的运行。MLOpen 或者 MATLABinit 则可以用来在原来的 Excel 进程中重新启动 MATLAB。

表 11-1　Excel Link 连接管理函数

函　数	作　用
MATLABinit	初始化 Excel Link，启动 MATLAB
MLAutoStart	自动启动 MATLAB
MLClose	终止 MATLAB 进程
MLOpen	启动 MATLAB 进程

2. 数据管理函数

Excel Link 提供了 9 个数据管理函数，实现 Excel 和 MATLAB 之间的数据拷贝，并可在 Excel 中执行 MATLAB 命令。它们概括在表 11-2 中。

表 11-2　Excel Link 数据管理函数

函　数	作　用
MATLABfcn	对于给定的 Excel 数据运行 MATLAB 命令
MATLABsub	对于给定的 Excel 数据运行 MATLAB 命令并指定输出位置
MLDeleteMatrix	删除 MATLAB 矩阵
MLEvalString	执行 MATLAB 命令
MLGetMatrix	向 Excel 数据表写 MATLAB 矩阵的数据内容
MLGetVar	向 Excel 数据表 VBA 写 MATLAB 矩阵的数据内容
MLAppendMatrix	向 MATLAB 空间添加 Excel 数据表的数据
MLPutMatrix	用 Excel 数据表创建或覆盖 MATLAB 矩阵
MLPutVar	用 Excel 数据表 VBA 创建或覆盖 MATLAB 矩阵

值得注意的是：用户能以数据表单元函数的形式或者宏命令的形式调用除 MLGetVar( ) 和 MLPutVar( )以外的所有数据处理函数，MLGetVar( )和 MLPutVar( )只能以宏命令的形式被调用。

11.2.4　Excel Link 应用实例

数据插值通常用来估计已知数据点之间某些数据点的数值，这一技术在数值分析、图像处理以及数据可视化操作中有着十分重要的应用价值。MATLAB 提供了很多数据插值函数，这些函数使得用户可以在较高的执行速度和有效的内存利用率的前提下进行数据的平滑插值。

下面是一个数据插值的具体实例。它利用 Excel 数据表管理、显示原始数据和插值数据，利用 Excel Link 实现数据的交换，执行 MATLAB 的插值函数，调用 MATLAB 的图形显示功能，显示插值数据的三维彩色曲面。

应用实例包含在 MATLAB 安装目录的 Exlink 子目录下，双击“exlisamp.xls”启动 Excel。

可以看到 6 个标签页。单击 Sheet3 的标签，可以看见 Excel 的内容，如图 11-8 所示。数据表中的 A5: A29、B5: B29 和 C5: C29 单元包含了测量得到的数据，要求的插值点分别在 E11: E30 和 F6: T6 两个单元中。

Microsoft Excel - ExliSamp.xls

文件(F) 编辑(E) 视图(V) 插入(I) 格式(O) 工具(T) 数据(D) 窗口(W) 帮助(H)

Arial

startmatlab putmatrix getmatrix evalstring

A48 =

	A	B	C	D	E	F	G	H	I	J	K	L
1	Data Interpolation											
2												
3	Original Data					Interpolated Values						
4	Time	Temp	Volume									
5	0.025	68.00	2504.08			Temp						
6	0.050	68.05	2535.07		Time	68.0	68.5	69.0	69.5	70.0	70.5	71.0
7	0.075	68.07	2562.91		0.025	2504.08	2638.15	2707.32	2750.09	2784.91	2851.19	2911.62
8	0.100	68.09	2575.74		0.05	2507.26	2635.76	2704.79	2746.66	2779.96	2846.35	2907.00
9	0.125	68.20	2606.16		0.075	2510.83	2633.45	2702.58	2743.62	2775.40	2841.84	2902.75
10	0.150	68.50	2628.58		0.1	2513.93	2631.34	2700.70	2740.99	2771.27	2837.66	2898.88
11	0.175	68.85	2681.38		0.125	2515.14	2629.60	2699.17	2738.77	2767.61	2833.83	2895.40
12	0.200	69.22	2712.06		0.15	2514.31	2628.58	2698.02	2736.99	2764.49	2830.38	2892.31
13	0.225	70.08	2767.52		0.175	2511.84	2628.88	2697.25	2735.66	2762.00	2827.31	2889.59
14	0.250	70.33	2815.54		0.2	2508.10	2629.91	2696.87	2734.79	2760.22	2824.68	2887.26
15	0.275	70.59	2824.37		0.225	2503.37	2631.32	2696.88	2734.37	2759.24	2822.57	2885.29
16	0.300	70.85	2873.65		0.25	2497.84	2632.93	2697.28	2734.42	2759.10	2821.05	2883.68
17	0.325	71.11	2882.20		0.275	2491.66	2634.64	2698.05	2734.91	2759.76	2820.23	2882.43
18	0.350	71.44	2896.49		0.3	2484.92	2636.35	2699.18	2735.86	2761.12	2820.16	2881.55
19	0.375	71.82	2902.07		0.325	2477.71	2638.00	2700.64	2737.22	2763.09	2820.81	2881.06
20	0.400	72.33	2920.04		0.35	2470.07	2639.54	2702.41	2739.01	2765.59	2822.11	2880.97
21	0.425	72.65	2929.35		0.375	2462.06	2640.93	2704.45	2741.19	2768.54	2823.98	2881.29
22	0.450	73.46	2934.23		0.4	2453.70	2642.15	2706.75	2743.75	2771.89	2826.33	2882.03
23	0.475	73.85	2938.55		0.425	2445.03	2643.15	2709.26	2746.67	2775.62	2829.13	2883.20
24	0.500	74.22	3012.93		0.45	2436.07	2643.94	2711.97	2749.92	2779.68	2832.32	2884.78
25	0.525	74.37	3099.12		0.475	2426.82	2644.48	2714.84	2753.48	2784.06	2835.88	2886.78
26	0.550	74.55	3130.01		0.5	2417.31	2644.77	2717.84	2757.32	2788.73	2839.78	2889.19
27	0.575	74.67	3179.24		0.525	2407.54	2644.80	2720.95	2761.44	2793.67	2844.01	2891.99
28	0.600	74.72	3180.71		0.55	2397.51	2644.56	2724.14	2765.79	2798.87	2848.55	2895.19
29	0.625	75.00	3184.15		0.575	2387.24	2644.05	2727.39	2770.37	2804.31	2853.38	2898.77
30					0.6	2376.71	2643.25	2730.67	2775.14	2809.97	2858.49	2902.71
31	Excel Link Functions											
32	1. Transfer original data to MATLAB.											
33	0	<== MLPutMatrix("Labels", A4:C4)										
34	0	<== MLPutMatrix("X",A5:A29)										
35	0	<== MLPutMatrix("T",B5:B29)										
36	0	<== MLPutMatrix("V",C5:C29)										

图 11-8 通过 Excel 向 MATLAB 传送数据，MATLAB 插值得到的数据

执行步骤如下：

（1）向 MATLAB 传送原始数据。

激活 A33 单元，按 F2 键和回车键，执行 Excel Link 函数 MLPutMatrix("Labels", A4:C4)，将 Time、Temp 和 Volume 传递给 MATLAB。

类似地执行 A34，A35 和 A36（按 F2 键和回车键），通过 Excel Link 函数将原始数据拷贝到 MATLAB 工作区间中。

此时，用户可以在 MATLAB 工作区间中查到已有的变量，方法是在 MATLAB 提示符下键入 whos 命令：

```
>> whos
  Name        Size                    Bytes Class


  Labels      1x3                      208 cell array
  T           25x1                     200 double array
  V           25x1                     200 double array
  X           25x1                     200 double array
Grand total is 92 elements using 808 byte
```

可见，在 MATLAB 工作空间已经有 4 个变量，分别为执行 A33、A34、A35 和 A36 单元格里的 Excel Link 命令所传送进来的。

（2）向 MATLAB 传送需要插值的点。

执行 A39 和 A40 里的函数，向 MATLAB 传送需要插值的点。

（3）利用 MATLAB 函数，执行数据插值操作。

执行 A43 单元格里的函数，利用 MATLAB 的二维插值函数生成需要插值点的数据。这里采用的插值函数是 griddata( )。

（4）转换得到的插值数据矩阵，并向 Excel 回传数据。

执行 A46 和 A411 的函数，将体积数据转置后拷贝到 Excel 环境中来，数据添加到单元 F7:T30 中，如图 11-9 所示。

（5）绘制插值区域的图形并标注。

执行 A50 的函数，MATLAB 以三维彩色曲面图的形式将插值数据显示出来，如图 11-9 所示。

从这个例子可以看出：可以利用 Excel 数据表管理、显示原始数据和插值得到的数据，利用 Excel Link 实现数据的交换，由 MATLAB 执行插值和图形显示功能，从而得到插值数据的三维彩色曲面。

注意：很多读者也许感觉通过简单的操作就完成了 MATLAB 和 Excel 的数据共享，其中 Excel Link 的操作过程实际上包含在上述步骤中。例如，A43 单元的操作实际上是执行如下指令：MLEvalString("[XI, TI, VI] = griddata(X,T,V,Xa,Ta, 'invdist')")。其中，griddata( )函数完成表面填充，即数据的插值。其余步骤请读者仔细理解。

除了数据插值外，还可以进行常见的回归和曲线拟合。可以通过在 ExliSamp.xls 中单击“Sheet1”选项卡中的例程进行理解。

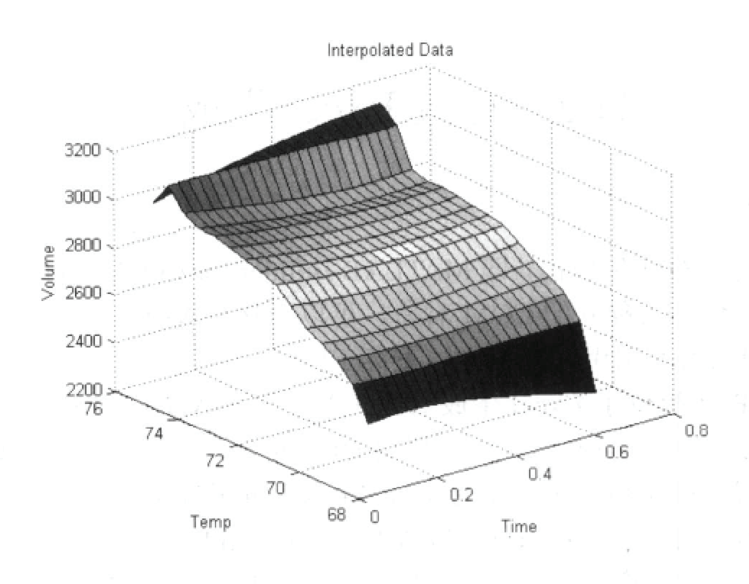


图 11-9　MATLAB 得到的插值数据图形

11.2.5　Excel Link 的注意事项

为了更有效地利用 Excel Link，下面对使用过程中要注意的一些细节问题进行说明。Excel Link 函数执行的是一个特定的操作，而 Excel 的函数返回一个确定的数值。所以，Excel 的操

作和函数在 Excel Link 函数的控制下会有不同的表现。

1. 大小写的区分

Excel Link 函数名对字母的大小写并不加以区分，例如 MLPutMatrix( )与 mlputmatrix( )代表的是同一个函数。而在 MATLAB 中对函数名是区分大小写的，且标准的 MATLAB 函数名通常使用小写字母，如 figure( )。

2. 参数的格式

Excel 的工作表等通常以“＋”或“＝”作为起始标记，例如：＝MLPutMatrix("Labels", A4:C4)。

3. 变量的定义方式

在大多数的 Excel Link 函数中有两种定义变量的方式：直接定义和间接定义。将变量用双引号标记即可直接定义变量，例如 MLDeleteMatrix(“Bonds”)，即将 MATLAB 中的矩阵变量 Bonds 删除。函数中不加双引号的工作区单元地址或行列名称被视为间接变量，函数对其指引内容进行操作。工作区单元地址可以包含页表序号，例如：

Sheet3! B1:C11，表示 Sheet3 表中 B1:C11 单元格中的数据。

4. 日期

默认的 Excel 日期从 1990 年 1 月 1 日开始，而 MATLAB 的日期是从 0000 年 1 月 1 日开始的。因此，在 MATLAB 计算中使用日期数字，将 Excel 日期转换为 MATLAB 日期时需要加上常量 693960。例如，1996 年 5 月 1 日在 Excel 中是 35200，而 MATLAB 中则是 729160，二者相差 693960。

11.3 通过 Excel 生成器

11.3.1 概述

MATLAB 提供了 COM 生成器，即 MATLAB Builder for COM，能把 MATLAB 开发的算法做成组件。这些组件作为独立的 COM 对象，可以直接被 Visual Basic、Visual C++或者其他支持 COM 的高级语言所引用。详细内容见第 12 章。

此外，MATLAB 还提供了 Excel 生成器的工具，即 MATLAB Builder for Excel。利用该工具，可以生成 DLL 组件和 VBA 代码。利用 DLL 组件，可以进行与 COM 生成器组件相类似的操作。VBA 代码则可以在 Excel 的 Visual Basic 编辑器中直接使用，可以保存为插件。

通过本节的学习，希望了解通过 MATLAB Excel Builder 创建、打包并且发布 Microsoft Excel 插件的方法，包括创建新的项目、添加文件创建 Excel/COM 接口、导入 VB 文件创建插件、测试插件和打包组件。

11.3.2 创建 Excel 生成器插件

在 MATLAB 命令行中输入命令“mxltool”创建工程，将显示 MATLAB 的 Excel 生成器主窗口，如图 11-10 所示。

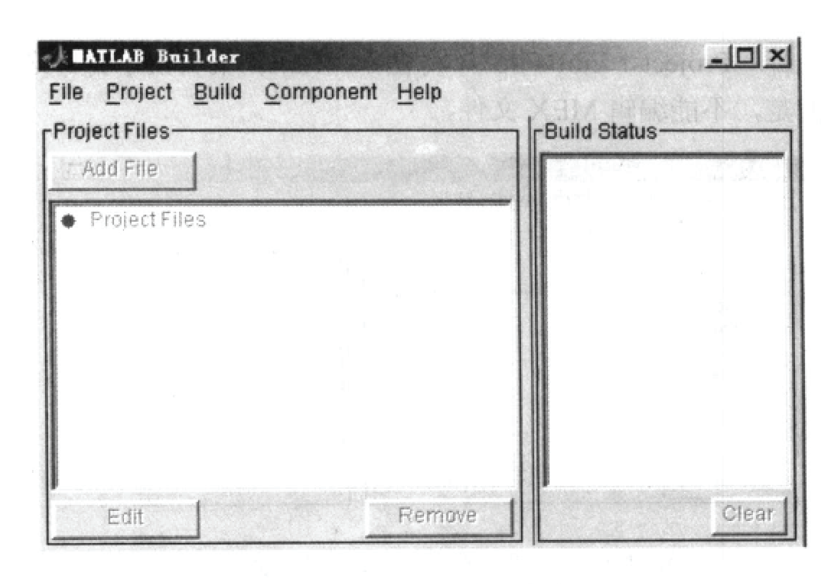


图 11-10　Excel 生成器主窗口

在窗口中选择 File，选择 New Project，打开 New Project Settings 对话框，如图 11-11 所示。在 Component name 文本框中输入组件的名称，在 Class name 文本框中输入类的名称。在 Project version 文本框中输入组件的版本号，默认版本号为 1。在 Project directory 文本框中输入工程目录，它指定编译和打包模型时将工程和生成的文件放在指定的目录中。在 Compile code in 方框选择是生成 C 代码还是 C++代码，用 C 代码写成的组件表现好，而 C++组件的可读性更强。

图 11-11　New Project Settings 对话框

创建工程后，主窗口中的“Project”、“Build”和“Component”等菜单选项变为可用，如图 11-12 所示。单击“Add File”按钮或者在菜单中依次选择“Project | Add File”，可在工程中添加 M 文件或 MEX 文件。单击“Remove”按钮或者在菜单中依次选择“Project | Remove File”，将删除选定的 M 文件或者 MEX 文件。单击“Edit”按钮或者双击 M 文件的文件名或

者在菜单中依次选择“Project | Edit File”，将在 MATLAB 编辑器中打开选定的 M 文件，进行修改和调试。但是，不能编辑 MEX 文件。

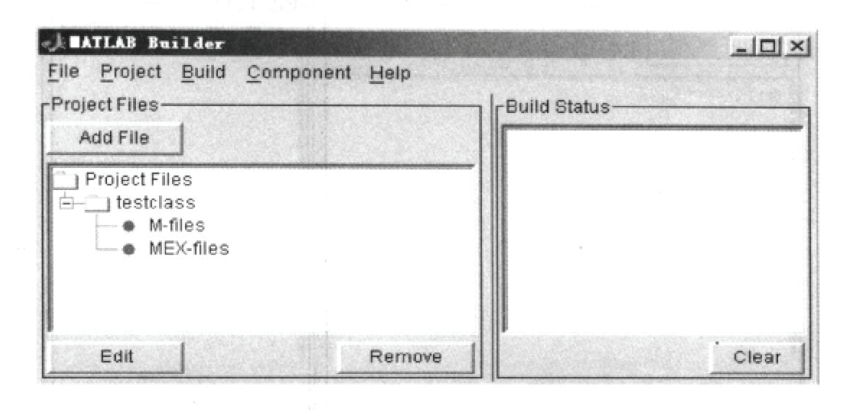


图 11-12 创建工程后的 Excel 生成器主窗口

在定义工程设置和添加必要的 M 函数和 MEX 函数后，可以生成一个可配置的 DLL 文件和必要的 VBA 代码。依次选择菜单选项“Build | Excel/COM Files”或者单击“Build”按钮，激活 MATLAB 编辑器。Builder Status 面板显示生成过程中的输出信息，并报告遇到的任何问题，例如 M 文件语法错误等。生成的结果是一个 DLL 文件和一个 VBA 文件（.bas）。生成的 DLL 会自动注册到系统中。依次选择菜单中“Build | Clear Status”或者单击“Clear”按钮，可以清除 Builder Status 面板的内容。生成过程中的输出信息会保存在 build.log 文件中，它可以通过依次选取“Build | Open Build Log”打开。该 Log 文件提供了一个生成过程的记录，在清除 Builder Status 面板的内容后还可以引用。

对于 VBA 模块的测试，可以通过在 Excel 的 Visual Basic 编辑器中导入 VBA 文件和激活 Excel 工作表中的函数来进行。要把 VBA 代码导入到 Excel 的 Visual Basic 编辑器中，方法是启动 Excel，并依次选取菜单选项“工具”、“宏”、“Visual Basic 编辑器”，然后在 Microsoft Visual Basic 窗口中，依次选择“文件”、“导入文件”，然后选择要打开的 VBA 文件。

由于 MATLAB 提供的 Excel 生成器的工具（即 MATLAB Builder for Excel）相对来说较为复杂，而且使用起来不如 Excel Link 直观、方便。尽管它生成的 VBA 代码可以直接在 Visual Basic 中使用，但应用还是较少。事实上，MATLAB 和 Visual Basic 的混合编程，完全可以通过其他的方式（例如 ActiveX、DDE 等）实现。因此，对 MATLAB 的 Excel 生成器，本节仅进行了简单的介绍。感兴趣的读者可以阅读 MATLAB 的帮助文件，方法是单击图 11-12 中 Excel 生成器主窗口的“Help”菜单，选择“MATLAB Builder for Excel”和“MATLAB Builder for Excel Help”子菜单，里面既包含了本节所介绍的基础知识，也包含了语法知识和一些例程。

11.4 直接将 MATLAB 工作区间的数据拷贝到 Excel

MATLAB 6.0 以后允许将 MATLAB 工作区间的数据拷贝到 Excel 中，以利用 Excel 的图表向导灵活地生成各种形式的图形，如折线图、散点图、饼图等。它在科研论文的写作中很实用。下面通过一个简单的实例来说明。

在 MATLAB 环境下键入以下命令，生成 t, y1, y2 等 3 个变量。

```
>> t=1:6;
>> y1=0.8*t-1;
>> y2=0.5*t+2;
>> plot(t,y1,'r+',t,y2,'b*');
```

MATLAB 执行后，会出现如图 11-13 所示的图形。

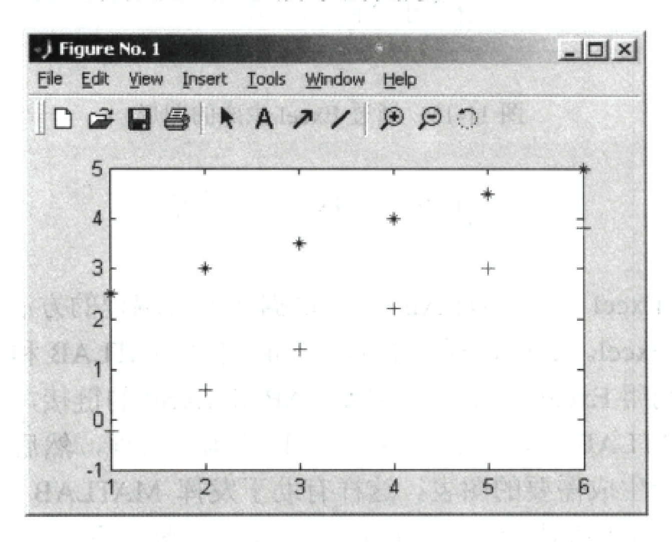


图 11-13　运行结果的 MATLAB 图形显示

将 MATLAB 工作空间中的变量导入 Excel 的方法是：单击 MATLAB 工作区的变量名，然后选中需要拷贝的数据，单击右键选择“复制”，然后在 Excel 中选中单元格后单击鼠标右键并选择“粘贴”。图 11-14 为从 MATLAB 工作区复制 y2 变量到 Excel 的过程。

复制 MATLAB 工作区中的数据到 Excel 后，选中刚复制过来的数据，再单击 Excel 工具栏的图表向导按钮，按照提示生成图表。图 11-15 为将 y1, y2 复制到 Excel 后按照默认设置生成的图形。

按照默认设置生成的图形可能不完全符合论文中图表的要求，还需要添加一些标题头、更改坐标刻度等。在 Excel 中，它们可以通过更改其属性灵活地实现。

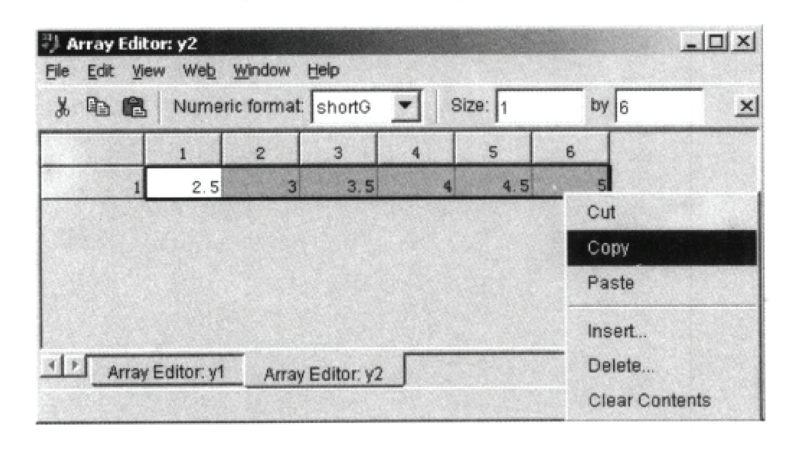


图 11-14　MATLAB 工作区中变量的复制

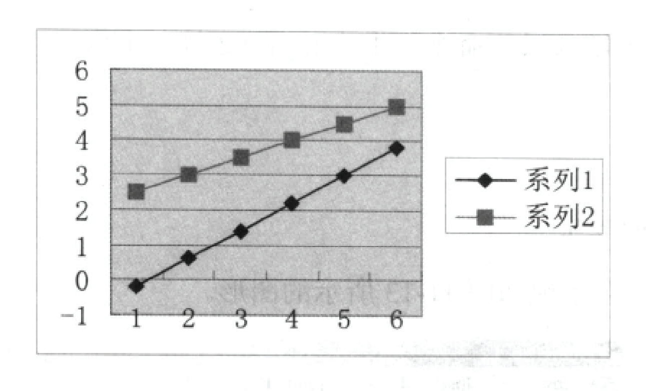


图 11-15　通过 Excel 生成的图形

11.5　小　　结

本章主要介绍了 Excel 和 MATLAB 共享数据和混合编程的方法，包括 Excel Link 和 MATLAB Builder for Excel。其中，通过 Excel Link 实现 MATLAB 和 Excel 链接的方法较为常见，应重点掌握。利用 Excel Link 实现 MATLAB 和 Excel 的链接，可以在 Excel 中将工作区中的变量导入到 MATLAB 中，并调用 MATLAB 的函数处理，然后将结果返加到 Excel 中处理，如利用图表向导生成需要的图表。这样有助于发挥 MATLAB 强大的数据处理能力，可以充分利用 Excel 的数据统计能力，生成需要的报表和图表等。

第 12 章　通过 MATLAB COM Builder 实现混合编程

12.1　COM 基础知识

组件对象模型（COM，Component Object Model）是 Microsoft 提出的以组件为发布单元的软件开发技术。COM 具有语言、平台无关的特性，可以方便软件的升级、定制与替换，是理想的软件应用方案。简言之，COM 是一种客户端/服务器（C/S）标准，提供了一类应用程序接口，允许任何符合标准的程序访问。

COM 为组件软件和应用程序之间进行通信提供了统一的标准，它为组件程序提供了一个面向对象的活动环境。COM 标准包括规范和实现两大部分，规范部分定义了组件和组件之间通信的机制，这些规范不依赖于任何特定的语言和操作系统，只要按照该规范，任何语言都可使用。COM 标准的实现部分是 COM 库，COM 库为 COM 规范的具体实现提供了一些核心服务。

由于市场等方面的原因，微软公司曾对由它自己推出的 COM 及其应用技术采用过不同的名称，从刚开始的对象链接与嵌入（OLE），到 ActiveX 控件及其自动化，再到组件对象模型 COM 及 COM+，因此造成了一些命名上的混乱，直接影响了很多读者的理解。尽管这样，这些名称所代表的模型及技术还是有一些细微的差别。

组件对象模型（COM）基本上是一项技术，它定义了一种标准的方式，使得客户端模块和服务器模块可以通过一个特殊接口进行通信。在这里“模块”表示应用程序或动态链接库，两个模块可以在相同的计算机中或通过网络在不同的计算机之间执行。目前，基本上所有基于 Windows 的开发工具都支持这项技术。例如，Microsoft Word，Excel 等。这些应用程序以自动化对象的方式暴露出内部的数据和功能，其他应用程序作为客户程序（或自动化控制器）可以访问这些自动化对象。人们所熟知的 DirectX 多媒体开发包也建立在 COM 组件技术之上。

COM 是面向对象的开发模型，COM 对象的概念同 C++对象的概念相似。面向对象的开发模型将数据和功能按照某种意义组合在一起形成类，对象就是类的实例。接口是 COM 对象与外界交流的唯一方式。因为 COM 对象是对现实世界中某种事物的抽象，所以其接口的所有函数在某种程度上有其相关性。COM 在 Windows 平台上以动态链接库的形式存在。采用动态链接库可以让 COM 组件客户决定什么时候加载所需要的 COM 组件。为了区分不同类别、不同开发商的 COM 组件，COM 规范采用一个 128 位长度的常量来标志 COM 组件。同样，为了唯一地标志组件的接口，COM 规范也采用这种 128 位 GUID 作为接口的标志。为了区分组件和接口的标志，一般将 COM 组件的 GUID 标志称为 CLSID，将接口的标志称为 IID。

显然，COM 作为不同语言之间的协作开发是非常方便的。COM 开发架构是以组件为基

础的，因而可以把组件看做用于“搭建”软件的积木块。采用这种开发模式，除了跨语言的特性以外，还可以带来很多好处，如采用组件替换使得软件系统的升级换代更加简单、可以在多个不同的软件开发应用中重复利用同一个组件等。

关于 COM 的更详细信息，请读者参考相关书籍。本章着重介绍 MATLAB 提供的 COM Builder 工具及其使用。

12.2 MATLAB 支持的组件自动化

MATLAB 支持组件自动化（COM Automation），即一个 COM 协议，该协议允许一个程序或组件去控制另一个程序或组件。因此，MATLAB 支持的组件技术可以分为以下三个方面的内容：

- 在 MATLAB 下运行其他软件的组件；
- 在其他程序下运行 MATLAB 的组件（包括 MATLAB 自身）；
- 将所需的 MATLAB 功能（通常由若干个 M 和 MEX 文件构成），利用 MATLAB 自带的 COM Builder 工具自动转换生成组件，以供其他程序使用。

其中，前两种方式易于理解，方便实用，但是必须是在安装有 MATLAB 的环境下才能实行，这不利于开发独立的应用程序。第三种方式使用 MATLAB COM Builder 将所需的 MATLAB 功能（通常由若干个.m 和. mex 文件构成）自动转换成一个组件库，供其他程序引用，可脱离 MATLAB 的环境运行。

12.2.1 在 MATLAB 下运行其他软件的组件

在 MATLAB 中运行其他软件的组件，主要用于扩展 MATLAB 功能，特别是那些 MATLAB 相对来说较弱的功能，例如访问硬件的能力等。实现时，以 MATLAB 为主，实现代码复用。MATLAB 支持 ActiveX。在 MATLAB 中调用 ActiveX 通常包含以下 4 个步骤。

1. 创建控制

```
h = actxcontrol(progid [, position [, fig_handle [, callback|
{event1 eventhandler1; event2 eventhandler2; ...} [, filename]]]])
```

其中，progid 是要生成的 ActiveX 控制的名称，由生产商提供。使用时，它必须与生产商提供的 PROGID 一致。position 是一个位置向量，指定 X，Y 坐标和宽、高信息，默认值为[20, 20, 60, 60]。fig_handle 是要生成 ActiveX 控制的图形窗句柄。回调函数 callback 是带可变数目参数的 M 函数名称。

2. 设置属性

可以通过 MATLAB 的 get 和 set 命令设置属性。它们的格式如下：

```
v = get(h[, 'propertyname'])
set(h, 'propertyname', value[, 'propertyname2', value2, ...])
```

其中，propertyname 为属性的名称，value 为属性值，h 是函数 actxcontrol( )返回的句柄。函数 set( )一次可以同时设置多个属性的值。此外，还可以通过函数 inspect( )查看属性值，并

进行必要的修改。

3. 触发方法

```
v = invoke(h, ['methodname' [, arg1, arg2, ...]])
```

函数 invoke( )用于触发，它的格式如下：

```
v = invoke(h, ['methodname' [, arg1, arg2, ...]])
```

其中，methodname 是一个字符串，它是要触发的方法的名称，argn 是所需要的触发事件的参数。

函数 methods ( )可用于返回所有的事件，methodview ( )则返回所有的事件及其变量名称。

4. 处理事件

函数 events( )显示 ActiveX 支持的所有事件，函数 eventlisteners( )则列出所有事件，以及回调函数或事件句柄。

下面给出两个 MATLAB 作为自动化客户端的实例。一个在 MATAB 环境中通过 ActiveX 调用 Windows Media Player，另一个是 MATLAB 自带的例子，把 MATLAB 作为自动化客户端，把 Excel 作为服务器。

【例 12-1】 在 MATLAB 命令窗键入如下代码：

```
m=figure;                    %图形窗口的句柄
%   MediaPlayer.MediaPlayer 是 Windows Media Player 的 PROGID。
h=actxcontrol('MediaPlayer.MediaPlayer.1', [0 0 300 300], m);
```

输出结果如图 12-1 所示，可见在 MATLAB 的图形窗口中启动了 Windows Media Player。它与在 Windows 环境下单独启动 Media Player 几乎完全一样。

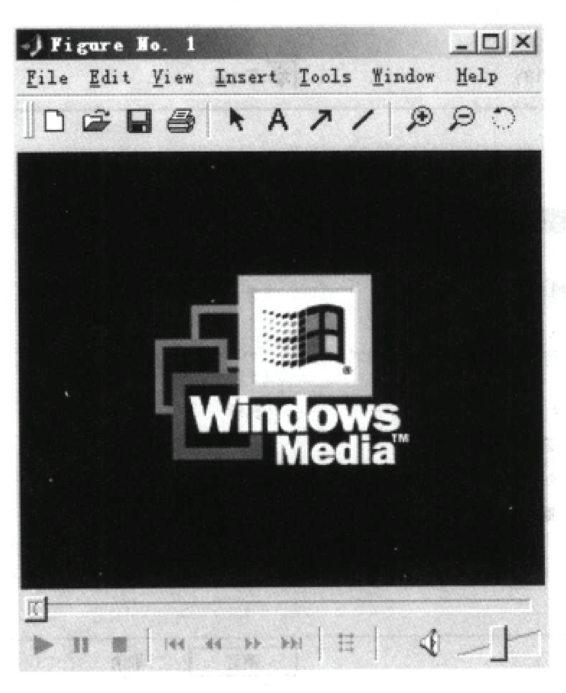


图 12-1　MATLAB 图形窗中启动 Media Player 的界面

为了播放视频文件，必须设置要播放的视频文件的名称。假设视频文件位于 F:\movies 目录下，文件名为 Aryan.mpg。为此，在 MATLAB 环境下键入以下语句：

```
set(h, 'filename', 'F:\movies\Aryan.mpg');
```

通过函数 set( )可以设置 Media Player 的属性值如图 12-2 所示。通过函数 propedit ( )查看属性值，如图 12-3 所示。

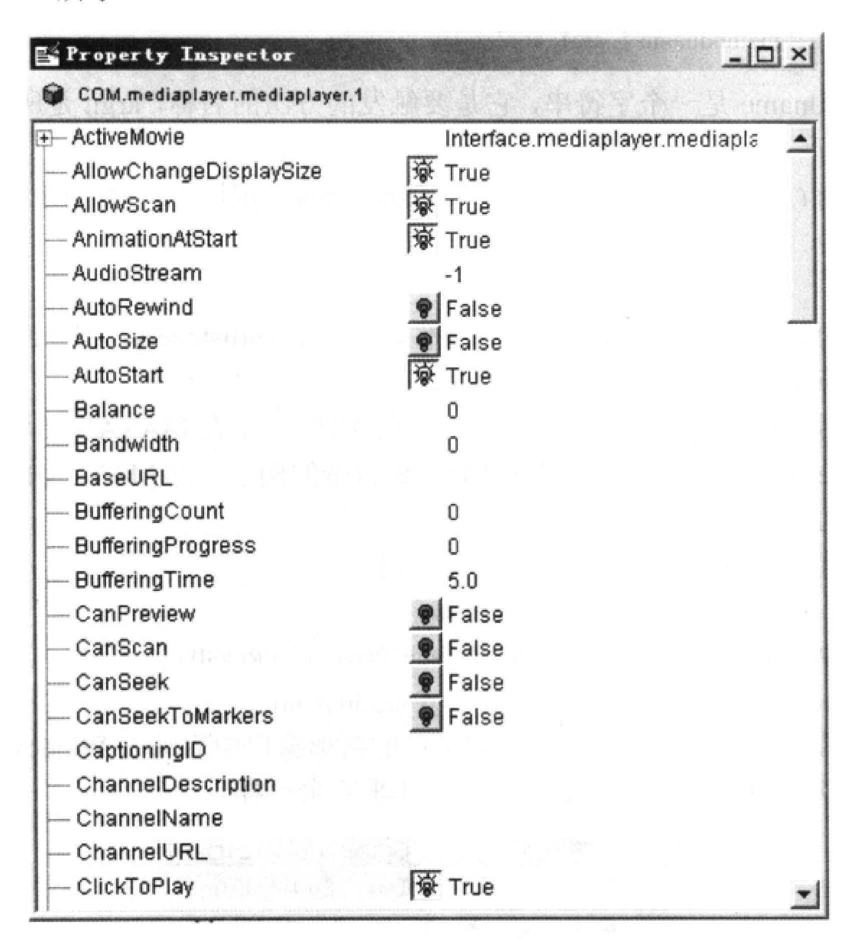


图 12-2　属性观察器

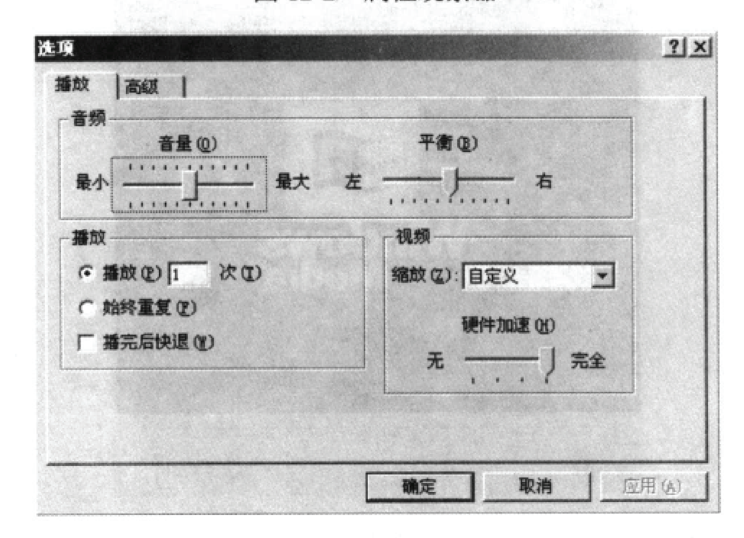


图 12-3　Media Player 播放属性的选项设置对话框

在 MATLAB 命令行键入下列命令可以启动视频文件的播放。其中，play 为要触发的播放事件名称。

```
>>invoke(h, 'play');
```

在 MATLAB 命令行键入“methods(h)”，可以列出 mediaplayer.mediaplayer.1 支持的所有事件，归纳于表 12-1 中。

表 12-1　Media Player 支持的 ActiveX 事件

AboutBox	GetMediaParameter	Previous	load
Cancel	GetMediaParameterName	SetCurrentEntry	move
FastForward	GetMoreInfoURL	ShowDialog	propedit
FastReverse	GetStreamGroup	Stop	release
GetCodecDescription	GetStreamName	StreamSelect	save
GetCodecInstalled	GetStreamSelected	addproperty	send
GetCodecURL	IsSoundCardEnabled	delete	set
GetCurrentEntry	Next	deleteproperty	invoke
GetMarkerName	Open	events	Play
GetMarkerTime	Pause	get	GetMediaInfoString

【例 12-2】以下是 MATLAB 作为自动化客户端调用 Excel 服务器的例程源代码，说明请参考源代码中的注释。它将 MATLAB 工作空间中的变量导入 Excel，程序的运行结果如图 12-4 所示。

```
% 首先，打开一个 Excel 服务器
h=actxserver('Excel.application');
% 插入一个新的工作簿
eWorkbooks=get(h, 'Workbooks');
eWorkbook=Add(eWorkbooks);
set(h, 'Visible',1);

% 激活第 2 个表
eActiveWorkbook=get(h, 'ActiveWorkbook');
eSheets=get(eActiveWorkbook, 'Sheets');
eSheet2=Item(eSheets, 2);
Activate(eSheet2);

% 获取表的句柄
eActiveSheet=get(h,'ActiveSheet');
% 把 MATLAB 数组导入 Excel
A=[1 2; 3 4];
eActiveSheetRange = Range(eActiveSheet, 'A1', 'B2');
set(eActiveSheetRange, 'Value', A);
```

```
% 返回一个范围。它是一个单元数组，因为单元范围可以包含不同类型的数据
eRange=Range(eActiveSheet, 'A1', 'B2');
B=get(eRange, 'Value');

% 转换为 double 矩阵，单元数组必须只包含标量
B=reshape([B{:}], size(B));

% 保存工作簿
SaveAs(eWorkbook, 'myfile.xls');
```

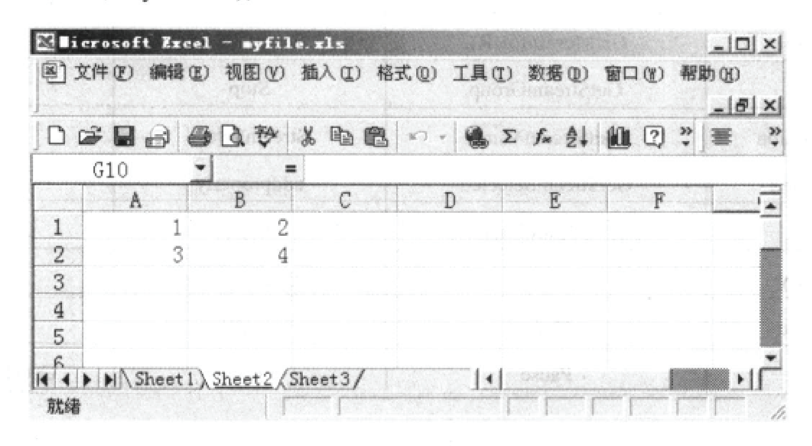


图 12-4　Excel 作为 MATLAB 客户端的服务器

12.2.2　在其他程序下运行 MATLAB 的组件

Microsoft 把所有以 COM 为基础的技术统称为 ActiveX 技术。自动化是大多数 ActiveX 技术的基础，它可使解释性的宏语言，例如 Visual Basic，能够在不了解应用程序实现细节的情况下控制自动化对象。MATLAB 实现了 ActiveX 自动化服务支持。MATLAB ActiveX 自动化服务的功能主要包括在 MATLAB 工作空间执行 MATLAB 命令，以及直接从工作空间存取矩阵等。下面列出了 MATLAB 自动化服务支持的几个主要方法，其参数和返回值的类型采用 ActiveX 自动化协议所定义的与语言无关的类型描述。

➢ BSTR Execute([in] BSTR Command)

BSTR 表示宽字符串类型，它与 Visual Basic 存储字符串所采用的数据格式相同。该方法接收字符串命令并在 MATLAB 中执行，将结果以字符串形式返回。

➢ void GetFullMatrix([in] BSTR Name，[in] BSTR Workspace，[in, out] SAFEARRAY (double)* pr，[in, out] SAFEARRAY(double)* pi);

该方法从指定的工作空间检索一个完整的一维或二维的实型或虚型 mxArray，其实部和虚部被分别存放到两个单独的 Double 型数组中。

➢ void PutFullMatrix([in] BSTR Name，[in] BSTR Workspace，[in] SAFEARRAY(double) pr，[in] SAFEARRAY(double) pi);

该方法将一个 mxArray 放入指定的工作空间。各参数的含义及调用方法与 GetFullMatrix 方法类似。

各种高级语言对组件技术的支持大同小异。现以 Visual Basic 为例说明通过 ActiveX 自动化接口可将 MATLAB 作为 Visual Basic 语言的一个 ActiveX 部件调用。部分示例代码如下：

```
Dim MATLAB As Object           '声明对象
Dim MReal1(5，5) As Double     '声明存放实部的 Double 型数组
Dim MImag( ) As Double          '声明存放虚部的 Double 型数组
Set MATLAB=CreateObject("MATLAB.Application")      '初始化对象
MATLAB.Execute("a=hilb(5)")                        '执行 MATLAB 命令
'将结果分别存入实部、虚部数组
Call MATLAB.GetFullMatrix("a","base"，MReal1，MImag)
```

12.2.3 MATLAB COM Builder 简介

2002 年，MathWorks 公司在其推出的 MATLAB 6.5 版本中新增了一个功能强大的 MATLAB COM Builder 模块。它提供了简单、易用的图形化用户界面，帮助用户将 MATLAB 的 M 函数文件自动、快速地转变为独立的进程内 COM 组件，它以.DLL（ActiveX DLL）形式被装入到客户的进程空间中，可以在任何支持 COM 组件的应用中使用，例如 Visual C++，Visual Basic，Microsoft Excel 等。

MATLAB COM Builder 编译的 COM 组件实现了标准的 IDispatch 接口，提供了对自动化（Automation）的支持。自动化对象公开了方法和属性，方法指自动化对象提供的功能服务，类似于类的成员函数；属性指自动化对象的数据特征，类似于成员变量。另外，为了实现组件对象与客户端的通信，增强了程序的交互性，MATLAB 组件也支持事件。事件是回调函数，由客户端实现，事件发生时，组件会自动调用实现的客户端事件函数。

应用组件技术实现 MATLAB 与其他高级语言混合编程，具有许多独特的优点，主要体现在以下几个方面：

- 适应性广，可用于 Visual C++、VB、Delphi、BC 等各种支持组件技术的高级语言中；
- 方便灵活，既可以与 MATLAB 协同工作，也可以生成不依赖 MATLAB 环境的独立程序，因此可获得最快的运行速度；
- 不需进行代码转换，使得编程风格一致，可读性好。

因此，本章着重介绍 MATLAB COM Builder，包括它的安装、配置 C/C++编译器以及在 Visual C++中使用 MATLAB COM Builder 生成的组件。

12.3 MATLAB COM Builder 使用

12.3.1 配置 MATLAB C/C++编译器

MATLAB COM Builder 是 MATLAB Compiler 的扩展，创建 COM 组件之前应安装 MATLAB 6.5 中的 MATLAB Compiler 和 MATLAB COM Builder 等模块。另外，还需对 MATLAB Compiler 进行必要的配置，方法为：在 MATLAB 命令窗口键入“mbuild –setup”，将出现选择编译器的提问，所列出的编译器包括计算机中已安装的各种 C/ C ++ 编译器，选择 Visual C++ 6.0。详细的配置过程与第 9 章中 MATLAB Add-in 的安装相同，此处不再重复。实际上，COM 组件在生成过程中是首先利用 mcc 命令调用 MATLAB 编译器，然后调用 Visual C++编译器。

12.3.2 使用 MATLAB COM Builder

用 MATLAB COM 生成器创建 COM 组件是一个简单的过程。具体的操作流程包括以下

4 个步骤：①启动 COM Builder，创建一个工程，设置相关属性；②增加所需的功能，即添加相应的. m 和. mex 文件，添加组件的属性和事件；③编译创建组件；④包装、注册和发布组件文件。

1. 创建工程

配置完 MATLAB 的 C/C++编译器后，在 MATLAB 命令行通过运行 comtool 命令启动 MATLAB COM Builder 图形用户界面，如图 12-5 所示。

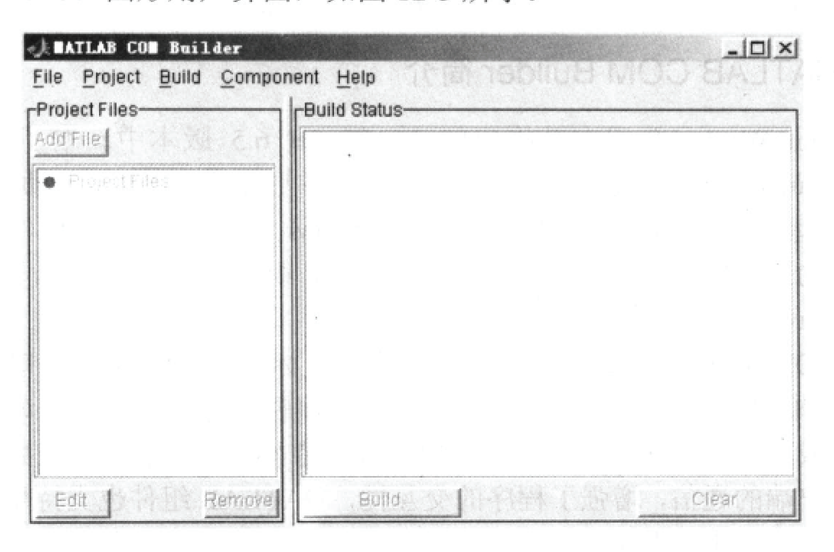


图 12-5　MATLAB COM Builder 图形用户界面

在 File 菜单中选择 New Project 选项，打开 New Project Setting 对话框，如图 12-6 所示。在 Component Name 文本框中输入组件（DLL 文件）的名称。输入组件名后，生成器将自动在 Class name 文本框中输入与组件名相同的名称。用户可以根据需要进行修改。尽管组件名和类名可以相同，但是必须注意组件名不能与任何 M 文件或 MEX 文件的文件名相同。

图 12-6　New Project Setting 对话框

若要把类添加到组件中，则在 Class name 文本框中输入类名，并单击“Add>>”按钮。添加的类将显示在“Classes”列表框中。在“Project Version”文本框中设置组件的版本号，默认的组件版本号为 1.0。在“Project directory”文本框指定在编译和打包模型时，工程和相关文件的存放位置。工程目录由当前目录和组件名自动组合生成。在“Compile code in”组合框中进行选择，可以设置生成 C 或者 C++代码。一般来说，用 C 代码编写的组件运行得好一些，而用 C++代码编写则可读性更强，便于修改和维护。在“Compiler Option”可以对编译过程进行选择。如果模型中包含 MATLAB 图形句柄调用，则通过选择“Use Handle Graphics Library”，在工程中包含 MATLAB C/C++图形库。如果选择“Build debug version”，则可以创建编译模型的一个调试版本，并能够在调用 MATLAB 编译器时指定详细输出。该调试版本一方面允许跟踪，使出错报告显示 M 文件和发生错误的行。所有跟踪信息都会进行报告。如果不进行调试，则得不到 MATLAB 代码中发生错误的位置和提示。另一方面，它允许 Visual Studio 调试器进行完整的调试。

设置完成后，单击“OK”按钮，即创建了一个工程（后缀为.cbl），工程文件被自动地添加到工程目录中。

2. 管理 M 文件和 MEX 文件

创建工程后，生成器的主窗口中的“Project”、“Build”和“Component”等 3 个菜单就变为可用。此时，单击“Add File”按钮可以添加 M 文件和 MEX 文件到工程中。一次只能添加一个文件。单击“Remove”按钮可以删除选定的 M 文件或 MEX 文件。选择 M 文件后单击“Edit”按钮，可以在 MATLAB 编辑器中打开该 M 文件并进行编辑和调试，但是不能编辑 MEX 文件。

M 函数文件既能够接收参数，也允许返回参数。另外，需要注意 M 函数文件的文件名必须和函数名一致；而且，MATLAB COM Builder 并不支持所有的 MATLAB 函数，如某些工具箱函数，具体限制可参阅 MATLAB 提供的编译器用户手册。

3. 生成组件

定义工程设置并添加必要的 M 函数和 MEX 函数后，可以通过“Build”按钮或者“Build”菜单中的“COM Object”选项来调用 MATLAB 编译器。中间源文件会写到工程目录的 src 子目录中，将必要的输出文件写到 distrib 目录中。

“Build Status”面板显示生成过程的输出，通知遇到的任何问题。生成 COM 过程完成后，在 distrib 子目录下会生成一个 DLL 文件。生成的 DLL 文件自动注册到系统中。选择“Build”菜单中的“Clear Status”选项，将清除“Build Status”面板内容。生成过程中会生成一个日志文件 build.log，它提供了生成过程的记录可以通过选择“Build”菜单中的“Open Build Log”选项打开。

4. 打包和分发组件

一旦模型编译成功并进行了测试，就可以把它打包并分发给终端用户。从“Component”菜单中选择“Package Component”选项，将创建一个自解压的可执行程序。它包含表 12-2 所示的文件。

表 12-2　自解压可执行程序包含的文件

文　　件	功　　能
_install.bat	由自解压可执行程序运行的脚本
<componentname_projectversion>.dll	编译后的组件
mglinstaller.exe	MATLAB 数学库和图形库安装器
mwcomutil.dll	COM 生成器工具库
mwregsvr.exe	在计算机上注册 DLL 的可执行程序

该自解压可执行文件后缀为.exe。在计算机上按以下步骤运行安装器：

- mglinstaller 安装 MATLAB C/C++数学库和图形库；
- 添加 mglinstaller 创建的目录；
- mwregsvr 注册 mwcomutil.dll 和<componentname_projectversion>.dll。

12.3.3　MATLAB COM Builder 工具库

MATLAB COM Builder 包含一组非常方便的数据转换工具库，包含了用于处理 MATLAB 特有的数据结构与 Visual C++客户交互的工具，例如结构（struct）、元组（cell）和稀疏矩阵等。这个工具库通过 COM 的形式提供给 Visual C++的用户，主要包括在 mwcomutil.dll 中，可以通过 DOS 下的 mwregsvr mwcomutil.dll 命令进行注册，也可以通过下面的办法直接使用：

```
#import "D:\matlab6p5\bin\win32\mwcomutil.dll" raw_interfaces_only
```

MATLAB COM Builder 工具库包含下面 7 个主要的类：

- MWUtil 类

对于 COM Builder 与客户而言，MWUtil 可以提供将 VARIANT 变量组成 VARIANT 数据，将 VARIANT 数组拆分为 VARIANT 变量以及 COM Builder 与 VARIANT 日期类型之间的转换等功能，由于只是利用了 MWUtil 的功能函数，在客户程序中最好存在一个全局范围内的 MWUtil 对象实例。

- MWFlags 类

它包含一组数组格式化和数据转换标志，所有的 MATLAB COM Builder 组件都包含一个 MWFlags 实例。通过修改 MWFlags，可以方便地改变对象的数据转换规则。

- MWStruct 类

使用 MWStruct 类，可以使调用 MATLAB COM Builder 的客户程序对 MATLAB 结构体进行操作。

- MWField 类

MWField 类包含了 MWStruct 类的字段信息的引用。MWField 类是不可创建的。

- MWComplex 类

使用 MWComplex 类，可以调用 MATLAB COM Builder 的客户程序对 MATLAB 的复数类型数值进行操作。

- MWSparse 类

使用 MWSparse 类，可以使调用 MATLAB COM Builder 的客户程序对 MATLAB 的稀疏矩阵数值进行操作。

➢ MWArg 类

MWArg 类用来向一个编译好的类传递参数，通过数据转换标志来实现将一个参数传入的目的。

12.3.4 在 Visual C++中调用 COM 组件的步骤

Visual C++调用 COM 组件的步骤如下：

（1）初始化 COM 库；

（2）得到 COM 对象的 CLSID；

（3）创建一个 COM 对象的实例；

（4）使用 COM 对象；

（5）退出 COM 库。

```
//初始化 COM 库
HRESULT CoInitialized(LPVOID pvReserved);    //Windows 保留参数，必须设为 NULL
//根据返回的 HRESULT 值来判断 COM 库的初始化是否成功
……
HRESULT hr;
hr=CoInitialize(NULL);
if (FAILED(hr))
{
return false;
}
……
//得到 COM 对象的 CLSID
CLSID CLSID_MyInterface;
HRESULT hr;
hr=CLSIDFromProgID(L"Myinterface.Myinterface1.0", &CLSID_MyInterface);
if (FAILED(hr))
{
MessageBox("CLSIDFromProgID 调用失败！");
return false;
}
//创建一个 COM 对象的实例
MyInterface *pIMyinterface
hr=CoCreateInstance(CLSID_MyInterface, NULL, CLSCTX_ALL, IID_ImyInterface, (void **)
&pIMyInface);  // IID_ImyInterface 接口 ImyInterface 的 IID
if (FAILED(hr))
{
MessageBox("创建 IMyInterface 接口失败！");
return false;
}
//使用创建的 COM 对象
```

```
....
退出 COM 库
CoUninitialize( );
```

12.4 在 Visual C++中使用 MATLAB COM Builder 生成的组件实例

下面通过一个实际例子来说明在 Visual C++中调用 MATLAB COM Builder 生成的组件。

1. 生成组件

（1）在 MATLAB 环境中，编辑以下三个 M 文件，分别命名为：myrandplot.m、getplotcolor.m 和 setplotcolor.m。

M 文件 myrandplot.m

```
function [out]=myrandplot(n)

global randplotcolor;      //为实现 COM 组件的属性，声明为 global 类型变量
if length(n)>1
     n=n(1);
end
out=rand(1, n);
if isempty(randplotcolor)
     randplotcolor=[0 0 1];
end
plot(out, 'color', randplotcolor);
M 文件 setplotcolor.m
function [ ]=setplotcolor(color)      //实现对 COM 组件属性的访问

global randplotcolor;
randplotcolor=color;
M 文件 getplotcolor.m
function [out]=getplotcolor( )       //实现对 COM 组件属性的访问

global randplotcolor;
out=randplotcolor;
```

（2）在 MATLAB 命令行通过运行 comtool 命令启动 MATLAB COM Builder 图形用户界面，然后选择 File 菜单下的 New Project 子菜单，创建一个新的工程，工程名为 myrandplottest。在弹出的对话框中完成相应的选项设置，包括在“compile code in”选项中选择 C，并且复选上“compiler options”里面的“Use Handle Graphics library”和“Build debug version”。这是因为前面的三个 M 文件中调用了 MATLAB 绘图指令，则需要 MATLAB 提供的 C/C++图形库，这时在编译器选项（Compiler Options）中必须选中“使用句柄图形库（Use Handle Graphics Library）”。如图 12-7 所示。

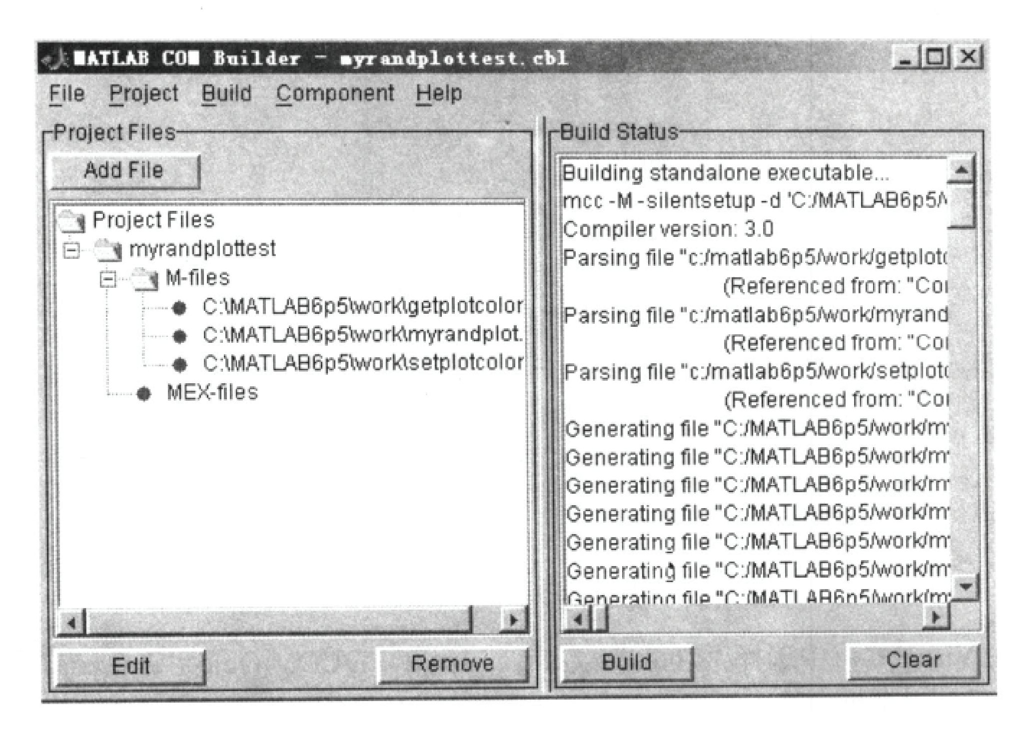


图 12-7　编译生成 COM 组件

（3）将 myrandplot.m、getplotcolor.m 和 setplotcolor.m 共三个 M 文件添加到工程中，并通过 Build 菜单下的 COM Object 子菜单编译当前工程的 COM 组件。待编译完成、生成 DLL 文件后，选择 “Component” 菜单下的“ Package Component”子菜单，将生成一个自解压的程序，并弹出如图 12-8 所示的进度对话框。在当前目录的 distrib 子目录下，包含表 12-2 所示的几个文件。

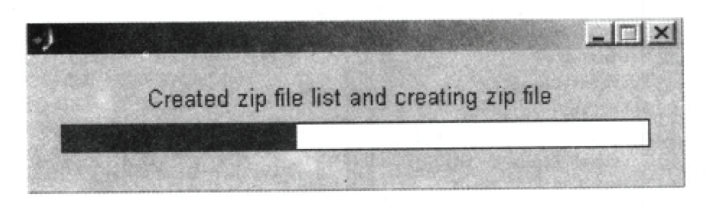


图 12-8　COM 组件打包过程的进度对话框

如图 12-7 所示，在编译生成组件的过程中，在“Build Status”面板生成了较多的信息。可以看出，COM 生成过程实际上是通过 MATLAB 编译器 mcc 生成的。其实，采用 mcc 和 mbuild 命令完全可以实现 MATLAB COM Builder 的全部功能，只不过 MATLAB COM Builder 为用户提供了一组更加友好的图形用户界面，省去了用户查阅 mcc 和 mbuild 用法中繁杂的选项，避免了出错。

2. 建立 Visual C++工程并设置必要的环境

（1）建立一个 Visual C++ MFC（exe）类型的工程，工程名为 myrandplottestvc，选择生成一个基于对话框的应用程序。删除原有的“确定”和“取消”按钮，并在对话框中添加两个 BUTTON，分别设置其 ID 为“ID_COMTEST”和“ID_EXIT”，CAPTION 属性分别设置为“MATLAB COM 测试”和“退出 COM”。调整对话框窗体布局如图 12-9 所示。

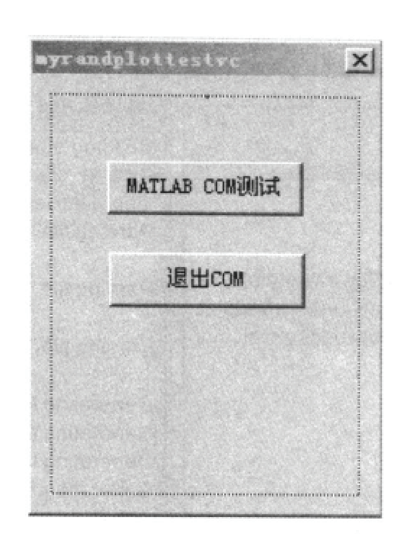


图 12-9　工程 myrandplottestvc 的对话框窗体

（2）在 Visual C++中选择“Tools”菜单，选择“OLE/COM Object Viewer”，将弹出如图 12-10 所示的对话框。选中新建的 MATLAB COM Builder 组件 myrandplottest，然后单击右键选择 View Type Information，将出现如图 12-11 所示的对话框。

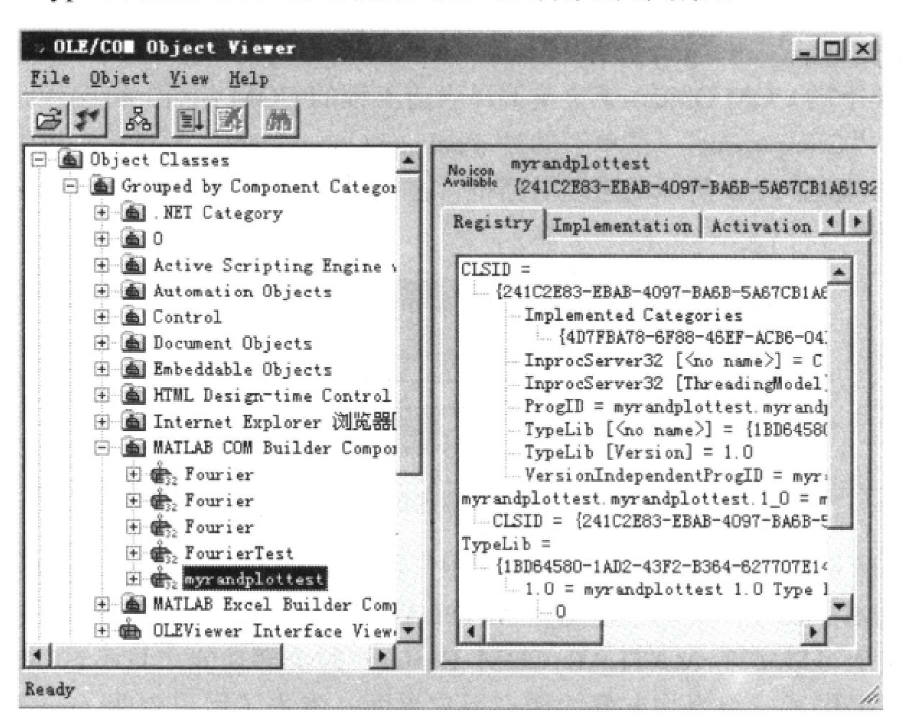


图 12-10　OLE/COM Object Viewer 对话框

（3）在 ITypeLib Viewer 对话框中，选择“File”菜单下的“Save As …”子菜单，然后分别选择*.h 和*.c 文件，将自动生成 myradplotttest_1_0.h 和 myrandplottest_1_0.c 文件，如图 12-12 所示。可见，生成的.c 文件名和.h 文件名是由工程名和版本号两部分组成。

（4）将生成的 myradplotttest_1_0.h 和 myrandplottest_1_0.c 文件拷贝到当前工程目录下，并通过“Project”菜单的“Add to Project”子菜单，将生成的 myradplotttest_1_0.h 和 myrandplottest_1_0.c 文件添加到工程中。

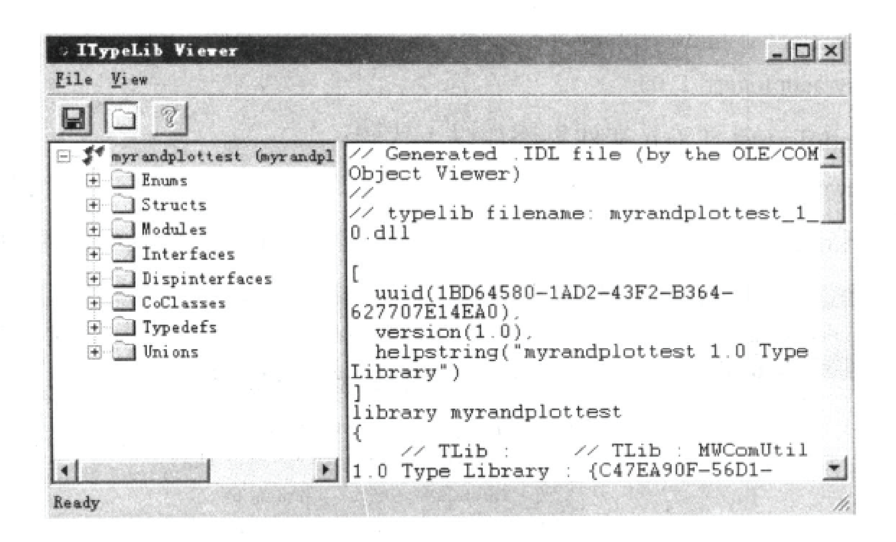


图 12-11　ITypeLib Viewer 对话框

另存为
保存在(I): distrib
bin
myrandplottest_1_0.c
文件名(N): myrandplottest_1_0
保存类型(T): C File with GUIDs (*.c)
保存(S)
取消

图 12-12　ITypeLib Viewer 保存程序对话框

（5）设置 Visual C++使用 MATLAB COM Builder 生成的 COM 组件需要的头文件路径。方法是通过“Tools”菜单的“Options”子菜单，选择“Directories”选项卡，在“Include files”中添加以下路径：

C:\matlab6p5\extern\include

C:\matlab6p5\extern\include\cpp

在“Library files”中添加以下路径：

C :\matlab6p5\extern\lib\win32

C:\matlab6p5\extern\lib\win32\Microsoft\msvc6

（6）通过“Project”菜单的“Setting”子菜单，选择“C/C++”选项卡，在“Category”下拉列表框中选择“Precompiled Headers”，如图 12-13 所示，选择“Automatic use of precompiled headers”，并设为“stdafx.h”。

3. 添加代码

（1）在 myrandplottestvcDlg.cpp 添加必要的头文件和库文件，即添加以下代码：

```
#include "mwcomutil.h"
```

```
#import "c:\matlab6p5\bin\win32\mwcomutil.dll" raw_interfaces_only
#include "myrandplottest_1_0.h"
```

（2）在 myrandplottest_1_0.h 文件里添加以下代码：

```
#include "mwcomutil.h"
```

（3）为 CmyrandplottestvcDlg 添加 public 类型的公共变量 Imyrandplottest *m_pTest。

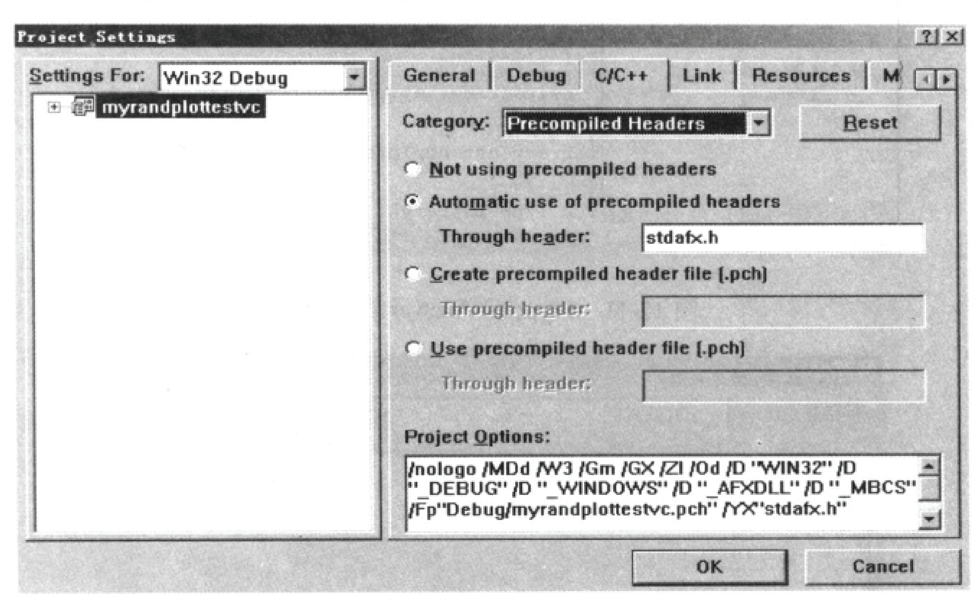


图 12-13　Visual C++工程 C/C++选项卡预编译头文件设置

（4）在对话框窗体的初始化函数 CMyrandplottestvcDlg::OnInitDialog( )中添加 COM 初始化代码：

```
// COM 初始化代码
m_pTest=NULL;

if ((FAILED(CoInitialize(NULL)))) {
    printf("COM  初始化失败!");
    exit(1);
}
```

（5）通过 ClassWizard 为对话框窗体中新添加的两个按钮添加响应代码：

```
void CMyrandplottestvcDlg::OnComtest( )
{
 // 以下代码为添加的
 HRESULT hr;
 Imyrandplottest *pTest=NULL;
hr=CoCreateInstance(CLSID_myrandplottest, NULL, CLSCTX_ALL, IID_Imyrandplottest, (void **)
&pTest);
 if (FAILED(hr)) {
     AfxMessageBox("创建 COM 出错!");
     return;
```

```
}

COleVariant in=100.0;
SAFEARRAY *pa;
double *pColor=NULL;
pa=SafeArrayCreateVector(VT_R8, 0,3);
hr=SafeArrayAccessData(pa, (void **)&pColor);
if (FAILED(hr)) {
      return;
}

pColor[0]=1.0;      //指定颜色，红、绿、蓝共 3 个分量
pColor[1]=0.0;
pColor[2]=0.0;
SafeArrayUnaccessData(pa);
VARIANT color;
color.vt=VT_R8|VT_ARRAY;
color.parray=pa;
pTest->setplotcolor(color);       //调用 COM 对象的属性
pTest->myrandplot(0, NULL, (VARIANT) in);
m_pTest=pTest;
}

void CMyrandplottestvcDlg::OnExit( )
{
 // 以下代码为用户添加的
 if (m_pTest !=NULL) {
      m_pTest->Release( );              //释放
      m_pTest=NULL;
 }

 OnCancel( );                          //退出应用程序
 CoUninitialize( );
}
```

此时，整个工程完成，可以编译本工程。运行界面如图 12-14 所示，单击“MATLAB COM 测试”按钮运行工程，结果是弹出 MATLAB 图形窗口 Figure No.1。单击“退出 COM”按钮则退出本工程。

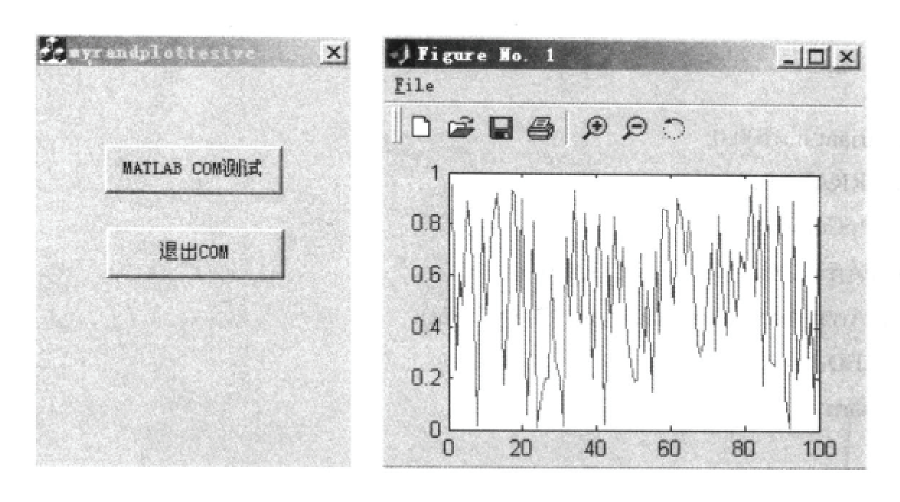


图 12-14　工程运行界面及结果

12.5　小　　结

本章介绍了 MATLAB 支持的组件自动化，尤其是通过 MATLAB COM Builder 生成 COM 组件，并通过一个工程实例介绍了生成的 COM 组件在 Visual C++环境下的使用。实际上，COM 生成过程实际上是通过 MATLAB 编译器 mcc 生成的。其实，采用 mcc 和 mbuild 命令完全可以实现 MATLAB COM Builder 的全部功能，只不过 MATLAB COM Builder 为用户提供了一组更加友好的图形用户界面。应用组件技术实现 MATLAB 与其他高级语言混合编程，具有适用于 Visual C++、VB、Delphi 、BC 等各种支持组件技术的高级语言、可以生成不依赖 MATLAB 环境的独立程序等优点，因此可获得最快的运行速度，不需进行代码转换，使得编程风格一致，可读性好。

第 13 章　混合编程综合应用实例

13.1　引　　言

前面各章介绍了 MATLAB 与其他高级语言混合编程的实例。为综合前面所学的知识，本章运用 MATLAB 混合编程，开发一个小型图像处理软件。图像处理系统中只包含了一些比较常见且相对较为简单的功能，目的在于演示 MATLAB 和高级语言混合编程的方法。希望感兴趣的读者，可以在本实例的基础上，继续完善其功能，使它真正成为一款优秀的图像处理软件。

本图像处理软件取名为 ImageProcessing。框架程序使用 Visual C++ 6.0 开发。为了充分体现 MATLAB 的混合编程功能，设计的图像处理软件中每一个图像处理功能分别给出了采用 C++及 MATLAB 的实现，然后通过接口连接到总的框架程序中。

13.2　预 备 知 识

13.2.1　数字图像处理简介

数字图像处理的英文名称是“Digital Image Processing”，指利用数字计算机或其他数字硬件，对图像信号进行某些运算，以提高图像的实用性。视觉信息是人类从大自然中获得的信息最主要的手段，在人类所获取的信息中，视觉信息约占 60%，而图像正是人类获取视觉信息的主要途径。数字图像处理技术的处理精度比较高，而且可以通过改进处理软件来优化处理效果。但是，数字图像处理的数据量非常庞大，因此对处理速度的要求也较高。

总的来说，数字图像处理通常包含以下内容：

➢ 点运算：针对图像的像素进行操作，可以有效地改变图像的直方图分布，有助于提高图像的分辨率以及图像均衡。

➢ 几何处理：包括图像的坐标变换、缩放、旋转、多个图像的配准以及图像的扭曲较正等，都是一些较基本的处理。

➢ 图像增强：指突出图像中的重要信息，减弱或去除不需要的信息。常用的有直方图增强以及伪彩色增强等。

➢ 图像复原：指去除干扰和模糊，从而恢复原图像，例如去噪声复原处理。

➢ 图像形态学处理：图像形态学是数学形态学的延伸，可以实现图像的膨胀、腐蚀、细化和分割等效果。

➢ 图像编码：指利用图像信号的统计特性及人眼视觉特性对图像进行高效编码，从而达到压缩图像的目的。

此外，还有图像重建、模式识别等。数字图像处理的应用越来越广泛，在工业生产、遥感技术、医学应用和安全等领域发挥着重要作用。

13.2.2 MATLAB 图像处理工具箱简介

从 MATLAB 5.3 版本开始，MATLAB 自带了数字图像处理的工具箱（Image Processing Toolbox）。最新的 MATLAB 7.0 版本，图像处理工具箱版本为 4.2。它提供了一套完整的用于图像处理和分析的函数，总共有 200 多个图像处理函数，功能强大。MATLAB 图像处理工具箱的函数功能归纳在表 13-1 中。另外，在 MathWorks 公司网站上提供了世界上 MATLAB 爱好者的文件交换区（MATLAB file exchange），包含了一些与图像和视频处理相关的函数，可以从网站 http://www.mathworks.com/matlabcentral/fileexchange/ 下载后放到 MATLAB 工作目录下，如同使用工具箱函数一样使用。图像处理工具箱与 MATLAB 的数据分析、算法开发和数据可视化环境集成在一起，可以帮助专业人士从耗时的图像处理算法编程中解脱出来，而把大部分时间用于问题本身的分析处理和算法研究上。因此，世界上数以万计的公司和大学都在使用 MATLAB 提供的图像处理工具箱，以解决各具挑战性的图像处理问题。

表 13-1　MATLAB 图像处理工具箱主要功能

主要功能	介绍	主要功能	介绍
图像显示	可控制显示单幅或多幅图像及其动画	多维信号处理	处理多维图像和数据，包括清晰化，滤波等
图像文件读取	从图像文件头中加载、保存或恢复图像信息，支持 BMP 、HDF 、JPEG、PCX、TIFF、PNG 和 XWD 等众多文件格式	线性滤波	对图像和 N 维数据块应用任意的或预先定义好的滤波器
几何操作	可对图像做不同的几何操作	线性 2D 滤波器设计	设计 2D 线性滤波器以满足给定的频域指标
像素值和统计	提取图像中像素的灰度统计值和其他信息	图像转换	对于高级分析选择不同的图像表达方法
图像分析	分析图像以获取其结构信息	相邻区域和块的处理	对图像进行相邻区域处理
图像增强	增强图像以使局部特征清晰可见或减少噪声	区域处理	对任意图像区进行处理和分析
空间变换	可以扭曲和矫正图像	二进制图像操作	计算大量的地貌和对象标记操作以增强图像及特征提取
图像配准	进行图像配准	颜色表（Colormap）操作	颜色表操作和使用少量颜色或替代颜色表近似表示索引图片（ Index image ）
色彩空间转换	把图像从一个色彩空间转换到另一个色彩空间	图像类型转换	把图像从一个数据类型转换到另一个数据类型

尽管使用 MATLAB 提供的图像处理工具箱进行图像处理方面的算法研究或工程设计编程效率极高，但是它不能脱离 MATLAB 环境运行，而且运行效率相对较低。因此，当图像处理算法研究和 MATLAB 仿真实验完成后，往往需要将算法改由其他的高级编程语言实现。Visual C++被认为是图像处理应用领域最理想的编程工具软件。正因于此，本章的综合实例

以 Visual C++为编程平台，期望通过与 MATLAB 的混合编程实现一个小的图像处理软件。

13.2.3 Visual C++的图像处理位图文件读/写操作

位图（BMP）文件格式是微软公司为其 Windows 环境设置的标准图像格式，而且它还包含了一系列支持 BMP 图像处理的 API 函数。随着 Windows 操作系统在世界范围内的不断普及，BMP 文件格式已经成为 PC 上最流行的图像文件格式之一。本章的综合实例也以 BMP 图像文件为例。因此，本节先简要介绍 BMP 文件格式。

Windows 中定义了两种位图文件类型，即一般位图文件格式和与设备无关的位图文件格式，其中设备无关位图（DIB）文件格式具有更强的灵活性与完整的图像数据。要进行图像处理，必须了解该图像文件的格式。Windows 中将 BMP 图像文件分为如下三个部分：文件头、调色板数据以及图像数据。

1. 文件头

文件头的长度为固定的 54 个字节，分为两个数据结构，其中一个数据结构中包含 BMP 文件的类型、大小和打印格式等信息，称为 BITMAPFILEHEADER；另外一个数据结构中则包含 BMP 文件的尺寸定义等信息，称为 BITMAPINFOHEADER。

BITMAPFILEHEADER 数据结构的定义为：

```
typedef struct tagBITMAPFILEHEADER
{
  WORD        bfType;
  DWORD  bfSize;
  WORD        bfReserved1;
  WORD        bfReserved2;
  DWORD  bfOffBits;
} BITMAPFILEHEADER;
```

这个结构的长度是固定的，为 14 个字节（WORD 为无符号 16 位二进制整数，DWORD 为无符号 32 位二进制整数）。

BITMAPINFOHEADER 数据结构的定义为：

```
typedef struct tagBITMAPINFOHEADER
{
  DWORD      biSize;
  LONG       biWidth;
  LONG       biHeight;
  WORD       biPlanes;
  WORD       biBitCount;
  DWORD  biCompression;
  DWORD  biSizeImage;
  LONG       biXPelsPerMeter;
  LONG       biYPelsPerMeter;
  DWORD  biClrUsed;
  DWORD  biClrImportant;
```

```
} BITMAPINFOHEADER;
```

它的结构长度也是固定的，为 40 个字节（LONG 为 32 位二进制整数）。

2. 调色板数据

这里是对那些需要调色板的位图文件而言的，真彩色图像是不需要调色板的，BITMAPINFOHEADER 后直接是位图数据。调色板实际上是一个数组，共有 biClrUsed 个元素（如果该值为零，则有 2 的 biBitCount 次方个元素）。数组中每个元素的类型是一个 RGBQUAD 结构，占 4 个字节，其定义如下：

```
typedef struct tagRGBQUAD
{
   BYTE  rgbBlue;      //该颜色的蓝色分量
   BYTE  rgbGreen;         //该颜色的绿色分量
   BYTE  rgbRed;           //该颜色的红色分量
   BYTE  rgbReserved;//保留字
} RGBQUAD;
```

3. 图像数据

对于用到调色板的位图，图像数据就是该像素颜色在调色板中的索引值，对于真彩色图像，图像数据就是实际的 R、G、B 值。下面就 2 色、16 色、256 色和真彩色位图分别介绍。对于 2 色位图，用 1 位就可以表示该像素的颜色（一般 0 表示黑，1 表示白），所以一个字节可以表示 8 个像素。对于 16 色位图，用 4 位可以表示一个像素的颜色，所以一个字节可以表示 2 个像素。对于 256 色位图，一个字节刚好可以表示 1 个像素。

13.3 综合实例框架

13.3.1 框架搭建

Visual C++包含编写程序源代码的文本编辑器、设计用户界面（菜单、对话框、图标等）的资源编辑器、建立项目配置的项目管理器、检查程序错误的集成调试器等工具，同时还提供了功能强大的应用程序向导 AppWizard 和类向导工具 ClassWizard。AppWizard 用于生成各种不同类型的、具有 Windows 界面风格的应用程序的基本框架，在生成应用程序框架后，使用 ClassWizard 便可轻松完成创建类、定义消息处理函数、重载虚拟函数等操作。

Visual C++的文档-视结构代表了一种新的程序设计方式，其核心是文档与视图的分离，即数据存放与显示（操作）的分离。文档-视结构大大简化了多数应用程序的设计开发过程。由于文档-视结构功能强大，因此一般用 Visual C++进行程序设计时都首先使 AppWizard 生成基于文档-视结构的单文档或多文档应用程序框架，然后在其中添加自己需要的代码，完成应用程序的特定功能。

ImageProcessing 程序的框架使用 Visual C++的 Appwinzard 向导工具搭建，具体创建步骤如下：

（1）启动 Visual C++ 6.0，从“File”菜单中选择“New”子菜单，单击“Project”标签。

选择创建 MFC AppWizard（exe）类型工程，如图 13-1 所示。工程名设定为 Dip，单击“OK”按钮。在“MFC AppWizard-Step 6 of 6”对话框中将 CDipView 类的基类改为“CscrollView”以支持视图滚动，如图 13-2 所示，其余各选项均采用默认值。按“Finish”按钮后，提示本工程创建的类的信息。单击“OK”按钮，MFC AppWizard 即创造了一个多文档程序的框架。

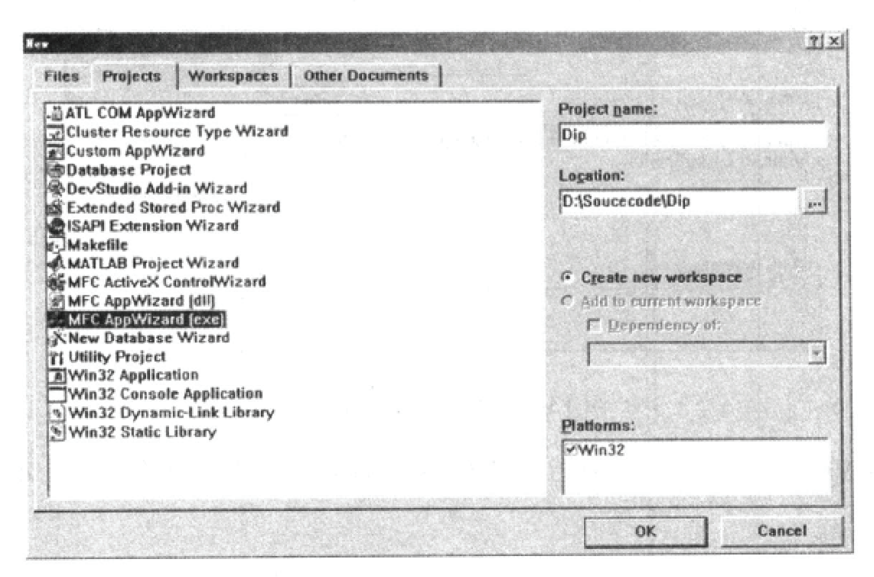


图 13-1　Visual C++ MFC AppWizard 向导

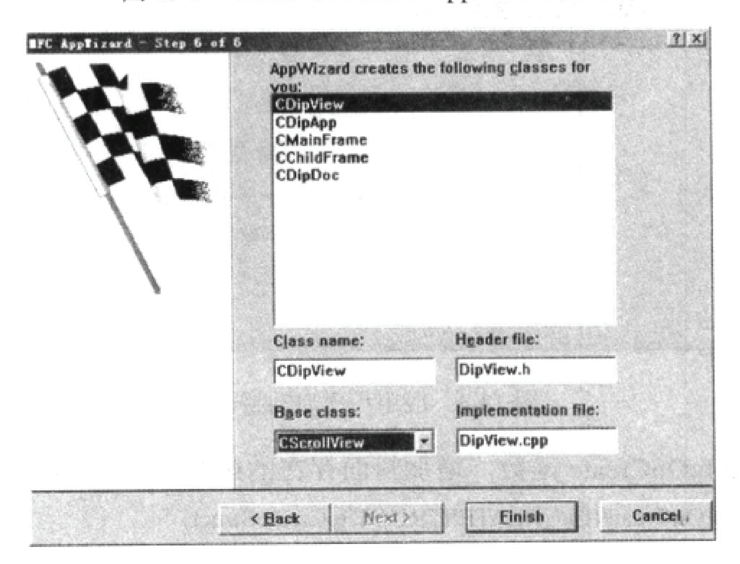


图 13-2　改变视 CDipView 的基类

（2）为应用程序添加 SplashScreen，如图 13-3 所示。一般的商业程序都有启动封面，我们也为 ImageProcessing 程序添加一个封面。从 Visual C++的“Project”菜单选择“Add To Project”子菜单，在“Components and Controls”对话框中，双击“Visual C++ Components”文件夹，然后选择“Splash Screen”组件，单击“Insert”按钮后就会为程序添加封面，同时在项目中会增加 CsplashWnd 类。

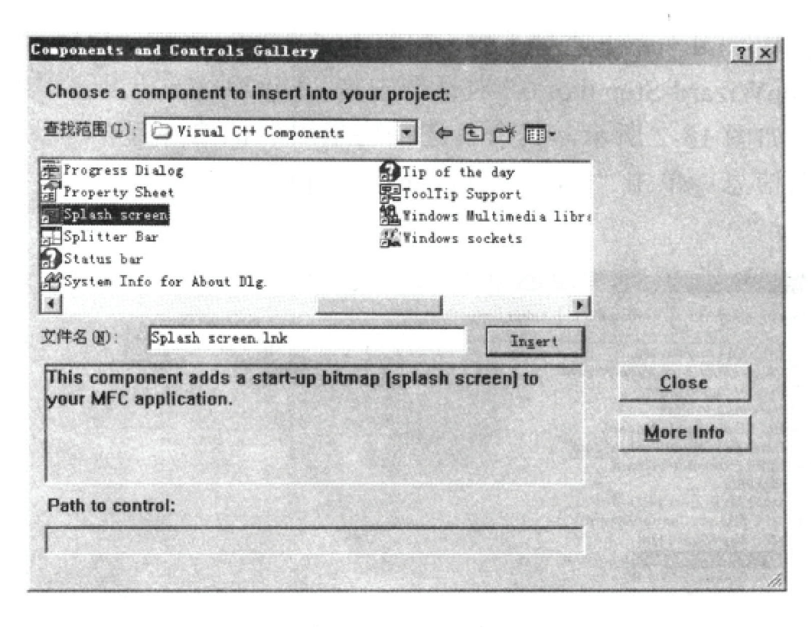


图 13-3　添加 Splash Screen

从 ResourceView 中选择 IDB_SPLASH 位图，编辑以设计自己的封面。本工程设计的封面如图 13-4 所示。



图 13-4　应用程序启动封面

修改 CSplashWnd::OnCreate 函数，增加封面存在的延时（改为 2s），如下所示：

```
int CSplashWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
  // Center the window.
        CenterWindow( );

        // Set a timer to destroy the splash screen.
  SetTimer(1, 2000, NULL);
  return 0;
}
```

13.3.2 模块划分

13.3.2.1 图像读取及转换 DLL

作为一个图像处理程序，必须能够读取多种格式的图像文件并进行图像处理。在ImageProcessing中，利用动态链接库ImageLoad.dll实现图像格式的读取以及各种格式之间的互相转换。它支持6种图像格式的读/写功能，分别为BMP格式、GIF格式、JPG格式、PCX格式、TGA格式和TIFF格式。利用Visual C++提供的Dependency Walker工具，可以查看ImageLoad.dll的导出函数列表，如图13-5所示。可见，该动态链接库共有22个函数，其中18个函数与图像文件的读/写操作有关，这些函数分为三类：Load函数用于读入图像文件；Save函数用于保存图像文件；Get函数用于获取图像文件的相关信息。对于ImageLoad.dll动态链接库，查看其定义头文件ImageLoad.h便可对其成员变量和成员函数有更全面的了解。

Dependency Walker - [ImageLoad.dll]

File Edit View Window Help

Ordinal	Hint	Function	Entry Point
1 (0x0001)	22 (0x0016)	_DllMain@12	0x00015D90
2 (0x0002)	0 (0x0000)	FileType	0x00015DC0
3 (0x0003)	1 (0x0001)	GetBMPInfo	0x000010E0
4 (0x0004)	2 (0x0002)	GetGIFInfo	0x00001640
5 (0x0005)	3 (0x0003)	GetJPGInfo	0x00013A90
6 (0x0006)	4 (0x0004)	GetLastPicLibError	0x00015DB0
7 (0x0007)	5 (0x0005)	GetPCXInfo	0x000168C0
8 (0x0008)	6 (0x0006)	GetTGAInfo	0x00017FF0
9 (0x0009)	7 (0x0007)	GetTIFInfo	0x00018AC0
10 (0x000A)	8 (0x0008)	LoadBMP	0x00001000
11 (0x000B)	9 (0x0009)	LoadGIF	0x00001280
12 (0x000C)	10 (0x000A)	LoadJPG	0x000137C0
13 (0x000D)	11 (0x000B)	LoadPCX	0x00016420
14 (0x000E)	12 (0x000C)	LoadTGA	0x00017750
15 (0x000F)	13 (0x000D)	LoadTIF	0x00018270
16 (0x0010)	14 (0x000E)	MedianCut	0x00015EC0
17 (0x0011)	15 (0x000F)	Popularity	0x00016EA0
18 (0x0012)	16 (0x0010)	SaveBMP	0x000011C0
19 (0x0013)	17 (0x0011)	SaveGIF	0x00001F40
20 (0x0014)	18 (0x0012)	SaveJPG	0x000135E0
21 (0x0015)	19 (0x0013)	SavePCX	0x000169B0
22 (0x0016)	20 (0x0014)	SaveTGA	0x000180B0
23 (0x0017)	21 (0x0015)	SaveTIF	0x00018C30

For Help, press F1

图 13-5 动态链接库的函数列表

头文件ImageLoad.h的详细内容如下（删除了部分注释信息）：

```
//Imageload.h
//////////////////////////////////// Generic ////////////////////////////////////


#define IMAGETYPE_NONE 0
#define IMAGETYPE_BMP 1
#define IMAGETYPE_GIF 2
#define IMAGETYPE_PCX 3
#define IMAGETYPE_TGA 4
#define IMAGETYPE_JPG 5
#define IMAGETYPE_TIF 6
```

```
#define IMAGETYPE_FIRSTTYPE IMAGETYPE_BMP
#define IMAGETYPE_LASTTYPE IMAGETYPE_TIF

#ifdef __cplusplus
extern "C" {
#endif

int __declspec (dllexport) GetLastPicLibError( void );

// Type filename
int __declspec (dllexport) FileType( const char * );

////////////////////////////////// BMP ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadBMP( const char * );
BOOL __declspec (dllexport) GetBMPInfo( const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SaveBMP( const char *, HGLOBAL );

////////////////////////////////// GIF ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadGIF( const char * );
BOOL __declspec (dllexport) GetGIFInfo( const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SaveGIF( const char *, HGLOBAL );

////////////////////////////////// PCX ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadPCX( const char * );
BOOL __declspec (dllexport) GetPCXInfo(const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SavePCX( const char *, HGLOBAL );

////////////////////////////////// TGA ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadTGA( const char * );
BOOL __declspec (dllexport) GetTGAInfo( const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SaveTGA( const char *, HGLOBAL );

////////////////////////////////// TIF ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadTIF( const char * );
BOOL __declspec (dllexport) GetTIFInfo( const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SaveTIF( const char *, HGLOBAL );

////////////////////////////////// JPG ////////////////////////////////
HGLOBAL __declspec (dllexport) LoadJPG( const char * );
BOOL __declspec (dllexport) GetJPGInfo( const char *, int *, int *, int *, int *, int * );
BOOL __declspec (dllexport) SaveJPG( const char *, HGLOBAL, int );
```

```
//////////////////////////////// PALETTE ///////////////////////////////
WORD __declspec (dllexport) MedianCut( WORD Hist[], BYTE ColMap[][3], int );
WORD __declspec (dllexport) Popularity( unsigned char *pBits, int nBits, int nWidth, int nHeight, BYTE ColorMap[][3] );

#ifdef __cplusplus
}
#endif
```

另外，为处理出错信息，还定义了一个 ImageErros.h 文件来声明读/写图像文件时的错误代码，其内容如下：

```
//ImageErrors.h

#ifndef __ERRORS_H__
#define __ERRORS_H__

#define IMAGELIB_SUCCESS 0

// File errors
#define IMAGELIB_FILE_OPEN_ERROR -1
#define IMAGELIB_FILE_CREATION_ERROR -2
#define IMAGELIB_FILE_WRITE_ERROR -3
#define IMAGELIB_FILE_READ_ERROR -4

// Memory erros
#define IMAGELIB_MEMORY_ALLOCATION_ERROR -50
#define IMAGELIB_MEMORY_LOCK_ERROR -51

// Argument errors
#define IMAGELIB_NODIB -100

#define IMAGELIB_UNSUPPORTED_FILETYPE -200
#define IMAGELIB_HDIB_NULL -201
#define IMAGELIB_LOGICAL_PALETTE_CREATION_ERROR -202
#define IMAGELIB_NO_PALETTE_FOR_HIGH_COLOR -203
#define IMAGELIB_STRETCHDIBITS_ERROR -204
#define IMAGELIB_PALETTE_QUANTIZE_ERROR -205
#define IMAGELIB_ATTEMPT_CHANGE_TO_SAME 206
#define IMAGELIB_ROTATION_VALUE_ERROR -207
#define IMAGELIB_ROTATE_ERROR -208
```

```
#endif
```

13.3.2.2 CDibObject 类封装

由于 MFC 没有封装处理与 DIB 位图相关的 Windows API 函数的类，所以基于面向对象的思想以及代码的重用性，有必要设计一个类来完成如下功能：

- 读取某种类型的图像文件；
- 获取图像的相关信息；
- 显示图像；
- 以相同或不同类型格式的图像文件保存该图像。

类的名字为 CDibObject，该类首先对 Imageload.dll 中的函数进行封装，然后添加一些代码实现图像的显示功能。类 CDibObject 的具体实现代码如下，其中包含了对代码注释。由于 DibObject.cpp 文件较长，在这里仅列举了几个关键函数。

```
//DibObject.h

#ifndef __DibObject_H__
#define __DibObject_H__

#include "ImageErrors.h"
#include "ImageLoad.h"

#define POPULARITY_PALETTE 0
#define MEDIAN_CUT_PALETTE 1
#define FIXED_PALETTE 2
#define GETRGB555( a, b, c, d ) { WORD *wData = (WORD *) d; a = (unsigned char) ( ( (*wData) & 0x7c00 ) >> 7 ); b = (unsigned char) ( ( (*wData) & 0x03e0 ) >> 2 ); c = (unsigned char) ( ( (*wData) & 0x001f ) << 3 ); }
#define GETRGB565( a, b, c, d ) { WORD *wData = (WORD *) d; a = (unsigned char) ( ( (*wData) & 0xf800 ) >> 8 ); b = (unsigned char) ( ( (*wData) & 0x07e0 ) >> 3 ); c = (unsigned char) ( ( (*wData) & 0x001f ) << 3 ); }
#define GETRGB888( a, b, c, d ) { DWORD *dwData = (DWORD *) d; a = (unsigned char) ( (*dwData) >> 16 ); b = (unsigned char) ( ( (*dwData) & 0x0000ff00 ) >> 8 ); c = (unsigned char) ( (*dwData) & 0x000000ff ); }
#define PUTRGB555( a, b, c, d ) { WORD *wData = (WORD *) d; *wData = ( ( ( (WORD) a & 0x00f8 ) << 7 ) | ( ( (WORD) b & 0x00f8 ) << 2 ) | ( (WORD) c >> 3 ) ); }
#define PUTRGB565( a, b, c, d ) { WORD *wData = (WORD *) d; *wData = ( ( ( (WORD) a & 0x00f8 ) << 8 ) | ( ( (WORD) b & 0x00fc ) << 3 ) | ( (WORD) c >> 3 ) ); }
#define PUTRGB888( a, b, c, d ) { DWORD *dwData = (DWORD *) d; *dwData = ( (DWORD) a << 16 ) | ( (DWORD) b << 8 ) | (DWORD) c; }
#define _RGB(r,g,b) (WORD)(((b)&~7)<<7)|(((g)&~7)<<2)|((r)>>3)

class CDibObject : public CObject
```

```
{
  DECLARE_DYNCREATE(CDibObject)
public:
  //无参构造函数
  CDibObject( );
  //拷贝构造函数
  CDibObject(CDibObject* pDibObject);
  //有参构造函数
  CDibObject( const char *, CDC *pDC = NULL, int nX = -1, int nY = -1 );
  //析构函数
  ~CDibObject( );
  //等号操作符重载
  void operator= (const CDibObject &DibObject);
  //读入图像
  BOOL Load( const char *, CDC *pDC = NULL, int nX = -1, int nY = -1 );
  //得到最近发生的错误码
  int GetLastError( void );
  //保存图像
  BOOL Save( const char *, int nType = -1 );
  //得到宽度（像素单位）
  int GetWidth( void );
  //得到高度（像素单位）
  int GetHeight( void );
  //得到每像素颜色位数
  int GetNumBits( void );
  //得到颜色数
  int GetNumColors( void );
  //得到调色板
  BOOL GetPaletteData( RGBQUAD * );
  RGBQUAD *GetPaletteData( void );
  //得到文件类型
  int GetImageType( const char * );
  int GetImageType( void );
  //得到图像信息
  BOOL GetImageInfo( const char *, int *pnWidth = NULL,
                     int *pnHeight = NULL, int *pnPlanes = NULL,
                     int *pnBitsPerPixel = NULL, int *pnNumColors = NULL );
  //绘制图像
  BOOL Draw( CDC *, int x = -1, int y = -1 );
  //设置调色板
  BOOL SetPalette( CDC * );
  //设置调色板类型
```

```
void SetPaletteCreationType( int );
//得到调色板类型
int GetPaletteCreationType( void );
//图像是否被载入
BOOL IsLoaded( void );
//扩展名索引
int ExtensionIndex( const char * );
//改变图像格式
BOOL ChangeFormat( int );
//得到调色板字节数
int GetPaletteBytes( void );
//得到位图句柄
HGLOBAL GetDib( void );
//得到调色板指针
CPalette *GetPalette( void );
//对输入的坐标进行调整，看是否超出图像大小
void NormalizeCoordinates( int *, int *, int *, int *, BOOL *bCompleteImage = NULL, BOOL
*bLessThanHalf = NULL );
//得到 DIB 位图内存指针
void *GetDIBPointer( int *nWidthBytes = NULL, int nNewBits = 0, int *nNewWidthBytes = NULL,
int nNewWidth = -1 );
//得到逻辑调色板
LOGPALETTE *GetLogPal( void );
//创建逻辑调色板
LOGPALETTE *CreateLogPalette( RGBQUAD *Palette, int NumColors );
//得到最近颜色索引
int GetNearestIndex( unsigned char, unsigned char, unsigned char, RGBQUAD *, int );
//设置质量
void SetQuality( int nQual ){ m_nQuality = nQual; }
//设置宽度
void SetWidth( int nWidth ){ m_nWidth = nWidth; }
//设置高度
void SetHeight( int nHeight ){ m_nHeight = nHeight; }
//设置平面数
void SetPlanes( int nPlanes ){ m_nPlanes = nPlanes; }
//设置平面数
void SetScreenPlanes( int nScreenPlanes ){ m_nScreenPlanes = nScreenPlanes; }
//设置颜色位数
void SetBits( int nBits ){ m_nBits = nBits; }
//设置颜色位数
void SetScreenBits( int nScreenBits ){ m_nScreenBits = nScreenBits; }
//设置颜色数
```

```
void SetColors( int nColors ){ m_nColors = nColors; }
//设置图像类型
void SetImageType( int nImageType ){ m_nImageType = nImageType; }
//设置最后错误类型
void SetLastError( int nLastError ){ m_nLastError = nLastError; }
//设置最后错误类型
void SetPaletteBytes( int nPaletteBytes ){ m_nPaletteBytes = nPaletteBytes; }
//设置 m_nX
void SetX( int nX ){ m_nX = nX; }
//设置 m_nY
void SetY( int nY ){ m_nY = nY; }
//获取 m_nPlans 值
int GetPlanes( ){ return m_nPlanes; }
//获取 m_nScreenPlans 值
int GetScreenPlanes( ){ return m_nScreenPlanes; }
//获取 m_nScreenBits 值
int GetScreenBits( ){ return m_nScreenBits; }
//获取 m_nQuality 值
int GetQuality( ){ return m_nQuality; }
//获取 m_nColors 值
int GetColors( ){ return m_nColors; }
//获取 m_nX 值
int GetX( ){ return m_nX; }
//获取 m_nY 值
int GetY( ){ return m_nY; }
//处理调色板
void ProcessPalette( void );
//字节宽度
int WidthBytes( int, int );
//设置位图
void SetDib( HGLOBAL hDib ){ m_hDib = hDib; }
//设置逻辑调色板
void SetLogPal( LOGPALETTE *pLogPal ){ m_pLogPal = pLogPal; }
//清除图像
void KillImage( void );
//处理图像头
void ProcessImageHeader( void );
static char *szExtensions[7];                    //扩展名
int m_nLastError;                                //最近错误
int m_nScreenPlanes, m_nScreenBits, m_nPaletteBytes;
//诊断调试函数
#ifdef _DEBUG
```

```
  virtual void Dump( CDumpContext &dc) const;
  virtual void AssertValid( ) const;
#endif
protected:
  //初始化参数
  void InitVars( BOOL bFlag = FALSE );
  //从位图创建调色板
  void CreatePaletteFromDIB( RGBQUAD *, int );
  LOGPALETTE *CreatePaletteFromBitmap( int, unsigned char *, int, int, int );
  RGBQUAD *MakePopularityPalette( int, unsigned char *, int, int, int );
  RGBQUAD *MakeMedianCutPalette( int, unsigned char *, int, int, int );
  RGBQUAD *MakeFixedPalette( int );
  int m_nWidth, m_nHeight, m_nPlanes, m_nBits, m_nColors, m_nImageType;
  int m_nX, m_nY;
  int m_nQuality;
  int m_nPaletteCreationType;
  CPalette m_Palette;
  HGLOBAL m_hDib;
public:
  char * GetImageName( );
  void ProcImgHead(void);
  char *m_pszFilename;
  LOGPALETTE *m_pLogPal;
};
#endif

//DibObject.cpp

#include "stdafx.h"
#include "DibObject.h"

IMPLEMENT_DYNCREATE(CDibObject, CObject)

char *CDibObject::szExtensions[] = { ".BMP", ".GIF", ".PCX", ".TGA", ".JPG", ".TIF", "" };

//Diagnostics and dump member functions, overridden
#ifdef _DEBUG
void CDibObject::Dump(CDumpContext &dc) const
{
  //call base class function first
  CObject::Dump(dc);
  //now do the stuff for our specific class
```

```
  dc<<"File Name:"<<m_pszFilename<<"\n";
}
#endif
#ifdef _DEBUG
void CDibObject::AssertValid( ) const
{
  //call inherited AssertValid first
  CObject::AssertValid( );
  //check CDibObject members...
  ASSERT(m_pszFilename != NULL); //must exist
  ASSERT(m_hDib != NULL); //must exist
}
#endif
////////////////////////////////////////////////////////////////////
//CDibObject 类的构造函数
//----------------------------------------------------------------------
//基本功能：这是一个无参数的构造函数。它简单地创建一个 CDibObject 对象并
//          初始化其内部变量
//----------------------------------------------------------------------
//参数说明：无
//----------------------------------------------------------------------
//返 回 值：无
////////////////////////////////////////////////////////////////////
CDibObject::CDibObject( )
{
  //调用辅助函数初始化其内部变量
  InitVars( );
}
////////////////////////////////////////////////////////////////////
//CDibObject 类的构造函数
//----------------------------------------------------------------------
//基本功能：用一个 CDibObject 对象通过复制操作来创建一个 CDibObject 对象
//----------------------------------------------------------------------
//参数说明：CDibObject *pDibObject
//----------------------------------------------------------------------
//返 回 值：无
////////////////////////////////////////////////////////////////////
CDibObject::CDibObject(CDibObject *pDibObject)
{
  InitVars( );
  KillImage( );
  m_nWidth = pDibObject->m_nWidth;
```

```
m_nHeight = pDibObject->m_nHeight;
m_nPlanes = pDibObject->m_nPlanes;
m_nBits = pDibObject->m_nBits;
m_nColors = pDibObject->m_nColors;
m_nImageType = pDibObject->m_nImageType;
m_nX = pDibObject->m_nX;
m_nY = pDibObject->m_nY;
m_nLastError = pDibObject->m_nLastError;
m_nScreenPlanes = pDibObject->m_nScreenPlanes;
m_nScreenBits = pDibObject->m_nScreenBits;
m_nPaletteBytes = pDibObject->m_nPaletteBytes;
m_nQuality = pDibObject->m_nQuality;
m_nPaletteCreationType = pDibObject->m_nPaletteCreationType;

int nNumColors = m_nColors;
int nWidthBytes = WidthBytes( m_nBits, m_nWidth );

if( pDibObject->m_hDib != NULL )
{
      DWORD dwSize = ::GlobalSize( pDibObject->m_hDib );
      char *pData = (char *) ::GlobalLock( pDibObject->m_hDib );
      if( pData != NULL )
      {
            HGLOBAL  hGlobal  =  ::GlobalAlloc(  GMEM_MOVEABLE  |  GMEM_ZEROINIT,
dwSize );
            if( hGlobal != NULL ){

                  char *pDestData = (char *) ::GlobalLock( hGlobal );
                  if( pDestData != NULL )
                  {
                        memcpy( pDestData, pData, dwSize );
                        ::GlobalUnlock( hGlobal );
                        m_hDib = hGlobal;
                  }
                  else ::GlobalFree( hGlobal );
            }
            ::GlobalUnlock( pDibObject->m_hDib );
      }
  }
}
////////////////////////////////////////////////////////////////
//CDibObject 类的构造函数
```

```
//------------------------------------------------------------------
//基本功能：构造一个 CDibObject 对象。唯一要求的一个参数是文件名。如果
//          给出了 CDC 设备上下文参数，图像加载后会立即在该设备上下文中显示
//          出来。如果给出了 nX 或 nY 参数，图像会显示在该坐标指定的位置，否
//          则，图像总是显示在坐标为（0, 0）的位置。
//------------------------------------------------------------------
//参数说明：const char *pszFilename
//          CDC *pDC，默认为 NULL
//          int nX，默认为-1
//          int nY，默认为-1
//------------------------------------------------------------------
//////////////////////////////////////////////////////////////////////
CDibObject::CDibObject( const char *pszFilename,
                        CDC *pDC, int nX, int nY )
{
  InitVars( );
  Load( pszFilename, pDC, nX, nY );
}
//////////////////////////////////////////////////////////////////////
//CDibObject 类的析构函数
//------------------------------------------------------------------
//基本功能：析构 CDibObject 对象，删除该对象中的图像及相应的变量
//------------------------------------------------------------------
//参数说明：无
//////////////////////////////////////////////////////////////////////
CDibObject::~CDibObject( )
{
  if( m_hDib ) ::GlobalFree( m_hDib );
  if( m_pszFilename != NULL ) delete [] m_pszFilename;
  if( m_pLogPal != NULL ) delete [] m_pLogPal;
}
//////////////////////////////////////////////////////////////////////
//BOOL Load( )
//------------------------------------------------------------------
//基本功能：本函数把一个图像文件载入 CDibObject 类。必需的一个参数是文件
//          名。如果给出了 CDC 设备上下文参数，图像一加载就被绘制。如果给
//          出了 nX 或 nY 参数，图像将显示在该坐标指定的位置。否则，图像总是
//          显示在坐标为（0,0）的位置。
//------------------------------------------------------------------
//参数说明：const char *pszFilename
//          CDC *pDC，默认为 NULL
//          int nX，默认为-1
```

```
//           int nY，默认为-1
//---------------------------------------------------------------------
//返 回 值：BOOL：成功返回 TRUE，失败返回 FALSE
////////////////////////////////////////////////////////////////////////
BOOL CDibObject::Load( const char *pszFilename,
                                    CDC *pDC, int nX, int nY )
{
  //获取图像文件类型
  m_nImageType = FileType( pszFilename );
  if( m_nImageType == 0 )
  {
        m_nLastError = IMAGELIB_UNSUPPORTED_FILETYPE;
        return( FALSE );
  }
  //删除已存在的图像
  KillImage( );
  m_pszFilename = new char [strlen(pszFilename)+1];
  if( m_pszFilename != NULL ) strcpy( m_pszFilename, pszFilename );
  //根据文件类型调用 ImageLoad.dll 动态链接库中的相应函数打开图像文件
  switch( m_nImageType )
  {
  case IMAGETYPE_BMP:
        m_hDib = ::LoadBMP( pszFilename );
        if( m_hDib == NULL ){
              m_nLastError = ::GetLastPicLibError( );
              return( FALSE );
        }
        break;
  case IMAGETYPE_GIF:
        m_hDib = ::LoadGIF( pszFilename );
        if( m_hDib == NULL ){
              m_nLastError = ::GetLastPicLibError( );
              return( FALSE );
        }
        break;
  case IMAGETYPE_JPG:
        m_hDib = ::LoadJPG( pszFilename );
        if( m_hDib == NULL ){
              m_nLastError = ::GetLastPicLibError( );
              return( FALSE );
        }
        break;
```

```
	case IMAGETYPE_PCX:
		m_hDib = ::LoadPCX( pszFilename );
		if( m_hDib == NULL ){
			m_nLastError = ::GetLastPicLibError( );
			return( FALSE );
		}
		break;
	case IMAGETYPE_TGA:
		m_hDib = ::LoadTGA( pszFilename );
		if( m_hDib == NULL ){
			m_nLastError = ::GetLastPicLibError( );
			return( FALSE );
		}
		break;
	case IMAGETYPE_TIF:
		m_hDib = ::LoadTIF( pszFilename );
		if( m_hDib == NULL ){
			m_nLastError = ::GetLastPicLibError( );
			return( FALSE );
		}
		break;
	}
	//处理图像信息头
	ProcessImageHeader( );
	//处理调色板
	ProcessPalette( );
	//若传入了设备上下文指针，就在指定的设备上下文中绘制图像
	if( pDC != NULL ) Draw( pDC, nX, nY );
	return( TRUE );
}
/////////////////////////////////////////////////////////////////////
//BOOL Save( )
//-------------------------------------------------------------------
//基本功能：本函数保存驻留于 CDibObject 对象中的当前图像（图像类型定义见
//		GetImageType( )函数
//-------------------------------------------------------------------
//参数说明：const char *pszFilename
//		int nType，默认为-1
//-------------------------------------------------------------------
//返 回 值：BOOL：成功返回 TRUE，失败返回 FALSE
//-------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////
```

```
BOOL CDibObject::Save( const char *pszFilename, int nType )
{
  //若没有指定文件类型，则根据文件名判断其文件类型
  if( nType == -1 ) nType = ExtensionIndex( pszFilename );
  if( nType < IMAGETYPE_FIRSTTYPE || nType > IMAGETYPE_LASTTYPE ) return( FALSE );
  m_nImageType = nType;
  delete [] m_pszFilename;
  m_pszFilename = new char [strlen(pszFilename)+1];
  if( m_pszFilename != NULL ) strcpy( m_pszFilename, pszFilename );
  //根据文件类型调用 ImageLoad.dll 动态链接库中的相应函数保存图像
  switch( m_nImageType )
  {
  case IMAGETYPE_BMP:
        return( ::SaveBMP( pszFilename, m_hDib ) );
        break;
  case IMAGETYPE_GIF:
        return( ::SaveGIF( pszFilename, m_hDib ) );
        break;
  case IMAGETYPE_JPG:
        return( ::SaveJPG( pszFilename, m_hDib, m_nQuality ) );
        break;
  case IMAGETYPE_PCX:
        return( ::SavePCX( pszFilename, m_hDib ) );
        break;
  case IMAGETYPE_TGA:
        return( ::SaveTGA( pszFilename, m_hDib ) );
        break;
  case IMAGETYPE_TIF:
        return( ::SaveTIF( pszFilename, m_hDib ) );
        break;
  }
        return( TRUE );
  }
////////////////////////////////////////////////////////////////
//BOOL Draw( CDC *pDC, int nX, int nY )
//---------------------------------------------------------------
//基本功能：本函数是在设备描述表上绘制图像。如果 X 和 Y 坐标没有给出，图像会
//          被画在(0,0)坐标或上次传入的有效坐标处
//---------------------------------------------------------------
//参数说明：CDC *pDC
//          int nX，默认为-1
//          int nY，默认为-1
```

```
//------------------------------------------------------------------------
//返 回 值: BOOL: 成功返回 TRUE, 失败返回 FALSE
////////////////////////////////////////////////////////////////////////////
BOOL CDibObject::Draw( CDC *pDC, int nX, int nY )
{
  if( nX != -1 ) m_nX = nX;
  if( nY != -1 ) m_nY = nY;
  m_nLastError = IMAGELIB_HDIB_NULL;
  //没有打开图像
  if( m_hDib == NULL ) return( FALSE );
  char *pTemp;
  //锁定图像对象句柄
  pTemp = (char *) ::GlobalLock( m_hDib );
  m_nLastError = IMAGELIB_MEMORY_LOCK_ERROR;
  if( pTemp == NULL ) return( NULL );
  //图像信息头
  BITMAPINFOHEADER *pBIH;
  pBIH = (BITMAPINFOHEADER *) &pTemp[sizeof(BITMAPFILEHEADER)];
  int nRet = ::StretchDIBits( pDC->m_hDC, m_nX, m_nY, m_nWidth, m_nHeight, 0, 0,
                    m_nWidth, m_nHeight, (const void FAR *) &pTemp[sizeof(BITMAPFILEHEADER)
+sizeof(BITMAPINFOHEADER)+m_nPaletteBytes], (BITMAPINFO *) pBIH, DIB_RGB_COLORS,
SRCCOPY );
  ::GlobalUnlock( m_hDib );
  m_nLastError = IMAGELIB_STRETCHDIBITS_ERROR;
  if( nRet != m_nHeight ) return( FALSE );
  m_nLastError = IMAGELIB_SUCCESS;
  return( TRUE );
}
```

13.3.3 应用程序功能添加

有了类 CdibObject 就可以在 13.2 节创建的应用程序框架基础上添加图像的显示、处理等基本功能。

将 DibObject.h 和 DibObject.cpp 两个文件复制到当前工程目录中。选择 Visual C++集成开发环境的“Project”菜单的“Add to project”子菜单，选择添加“Files...”选项，系统打开如图 13-6 所示的“Insert Files into Project”对话框。选择文件 DibObject.h 和 DibObject.cpp，单击“OK”按钮将这两个文件添加到当前工程中。这样便可以使用类中提供的方法和属性。由于类 CDibObject 中用到了 ImageLoad.dll 动态链接库，如前所述，先将与 ImageLoad.dll 动态链接库相关的 ImageLoad.h、ImageLoad.lib 和 ImageErros.h 复制到当前工程目录，并将 ImageLoad.lib 库文件添加到工程中。在要用到 ImageLoad.dll 的 C++头文件（.h）或实现文件（.cpp）中注意添加相应的头文件，方法是文件开头添加以下两条语句：

```
#include "ImageErros.h"
```

```
#include "ImageLoad.h"
```

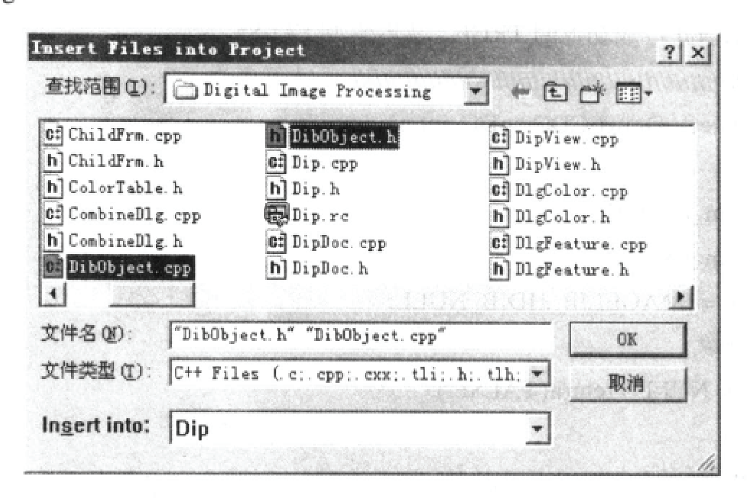


图 13-6 添加文件对话框

图像读取、显示和存储等基本功能实现方法如下：

1. 添加 OnFileOpen 函数以打开图像文件

应用程序类 CdipApp 自带了"文件\打开"菜单，在此为这个菜单项添加消息映射处理函数。使用 MFC ClassWizard 为 ID_FILE_OPEN 菜单命令消息添加 OnFileOpen 函数。

声明字符数组 szFilter，szFilter 保存了文件打开和关闭对话框的过滤器数组。在 Dip.cpp 文件头部添加如下代码：

```
char szFilter[] = "位图文件(*.BMP)|*.BMP|图形交换格式文件(*.GIF)|*.GIF|PCX 文件(*.PCX)|*.PCX|TGA 文件(*.TGA)|*.TGA|JPEG 文件(*.JPG)|*.JPG|标记图像文件(*.TIF)|*.TIF|所有支持图片|*.BMP,*.GIF,*.PCX,*.TGA,*.JPG,*.TIF|所有文件(*.*)|*.*||";
```

CdipApp::OnFileOpen 函数的代码如下：

```
void CDipApp::OnFileOpen( )
{
  static int nIndex = 1;

  CFileDialog FileDlg( TRUE, NULL, NULL, OFN_HIDEREADONLY, szFilter );
  FileDlg.m_ofn.nFilterIndex = (DWORD) nIndex;

  if( FileDlg.DoModal( ) == IDOK )
  {
       CString PathName = FileDlg.GetPathName( );
       PathName.MakeUpper( );
       OpenDocumentFile( PathName );
       nIndex = (int) FileDlg.m_ofn.nFilterIndex;
  }
}
```

2. 显示打开的图像

具体的图像绘制任务是在 CDipView 视图类成员函数 CDipView:: OnDraw 中完成的，其代码如下：

```
void CDipView::OnDraw(CDC* pDC)
{
  CDipDoc* pDoc = GetDocument( );
  ASSERT_VALID(pDoc);
  // TODO: add draw code for native data here
  if( !pDoc->m_bImageLoaded)
  {
        pDoc->LoadImageToDocument( );
  }

  //滚动窗口
  CSize sizeTotal;
  sizeTotal.cx = pDoc->m_pDibObject->GetWidth( );
  sizeTotal.cy = pDoc->m_pDibObject->GetHeight( );
  SetScrollSizes (MM_TEXT, sizeTotal);

  //获取客户区尺寸
  OnPrepareDC(pDC);
  CRect Rect;
  GetClientRect( &Rect );

  //获取图像宽度及高度
  int nImageWidth, nImageHeight;
  nImageWidth = pDoc->m_pDibObject->GetWidth( );
  nImageHeight = pDoc->m_pDibObject->GetHeight( );

  //当图像实际尺寸小于窗口尺寸时，将图像放在客户区中间显示
  int nX, nY;
  if( nImageWidth < Rect.Width( ) )
        nX = (Rect.Width( ) - nImageWidth ) / 2;
  else
        nX = 0;
  if( nImageHeight < Rect.Height( ) )
        nY = (Rect.Height( ) - nImageHeight ) / 2;
  else
        nY = 0;

  if( GetFocus( ) == this )
```

```
        pDoc->m_pDibObject->SetPalette(pDC);

    pDoc->m_pDibObject->Draw(pDC, nX, nY);

}
```

3. 保存打开的图像

具体的图像存储功能是由 CdipDoc 类成员函数 CDipDoc::OnFileSaveAs 实现的，本质上还是调用类 CDibObject 的 Save 函数，CDipDoc::OnFileSaveAs 函数的代码如下：

```
void CDipDoc::OnFileSaveAs( )
{
  // TODO: Add your command handler code here
  static int nIndex = 1;

  CFileDialog DialogSaveAs( FALSE, NULL, m_pDibObject->GetImageName( ),
                            OFN_HIDEREADONLY, szFilter );

  DialogSaveAs.m_ofn.nFilterIndex = (DWORD) nIndex;

  if( DialogSaveAs.DoModal( ) == IDOK )
  {
      CMainFrame *pMainFrame = ( CMainFrame * )AfxGetMainWnd( );
      CChildFrame *pChildFrame = ( CChildFrame * )pMainFrame->MDIGetActive( );
      CDipView *pDipView = ( CDipView * )pChildFrame->GetActiveView( );

      nIndex = (int) DialogSaveAs.m_ofn.nFilterIndex;
      if( nIndex == 5 )
      {
          if( m_pDibObject->GetNumBits( ) != 24 )
          {
              AfxMessageBox("必须是 24 位真彩色图像才能存为 JPEG 格式！");
              return;
          }
      }

      if( m_pDibObject != NULL )
      {
          CString strPathName = DialogSaveAs.GetPathName( );
          int nFindIndex = strPathName.Find(".");
          if( nFindIndex != -1)
              strPathName = strPathName.Left( nFindIndex );
                strPathName += CDibObject::szExtensions[ nIndex - 1 ];
```

```
                //m_pDibObject->ProcessImageHeader( );
                //m_pDibObject->ProcessPalette( );
                m_pDibObject->Save( strPathName );

                CString strFileName = DialogSaveAs.GetFileName( );
                nFindIndex = strFileName.Find(".");
                if ( nFindIndex != -1 )
                    strFileName = strFileName.Left( nFindIndex );
                strFileName += CDibObject::szExtensions[ nIndex - 1 ];
                pChildFrame->SetWindowText( strFileName );

                SetPathName( strPathName );
                if( nIndex == 5 )
                {
                    m_pDibObject->Load( strPathName );
                    pDipView->InvalidateRect( NULL, FALSE );
                    pDipView->UpdateWindow( );
                }
            }
        }
    }
```

至此读入、显示和保存一幅图像的代码就完成了。现在就可以编译、运行此程序并打开一幅图像来看看运行效果了。图 13-7 为打开 Image 目录下的 Miss.bmp 文件的效果。



图 13-7 程序打开图像时界面

13.4 实现方法

本节将在 13.3 节实现的 ImageProcessing 程序框架基础上添加一些图像处理功能。每种图像处理方法都附有简单的原理介绍、C++实现方法以及 MATLAB 混合编程实现方法。

通过本节的介绍，相信读者对混合编程会有更进一步的理解，能更加灵活地运用这些混合编程的方法。为了使图像处理功能有一个统一的界面，在编制 M 函数时不采用默认的 Figure 界面，去掉默认菜单和工具条。

Figure 的具体属性如下：

```
figNumber=figure( ...
        'Visible','off', ...
        'NumberTitle','off', ...
        'MenuBar','none');
```

Visual C++集成开发环境设置：

由于混合编程过程中用到了 MATLAB 定义的头文件以及库文件，在混合编程之前需要对编译环境进行设置。单击“Tools”菜单下的“Options...”子菜单，在弹出的“Options”对话框中选择“Directories”栏，添加头文件和库文件的路径，分别如图 13-8 和图 13-9 所示。单击“Project”菜单，选择“Settings”子菜单，在“Link”选项卡的“Object/library module”栏添加要用到的库文件名，如图 13-10 所示。

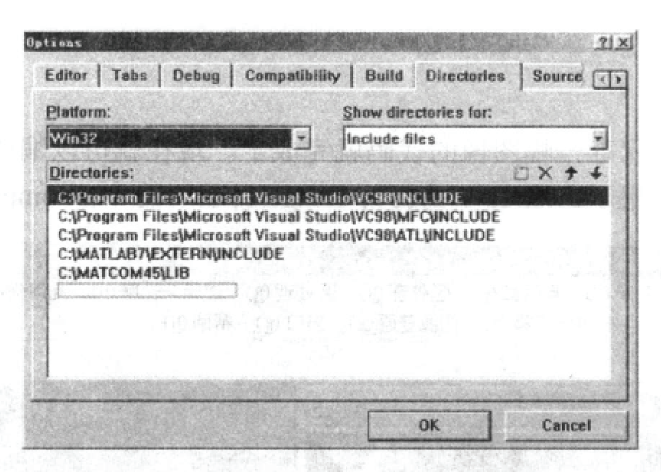


图 13-8　设置 Include files 目录

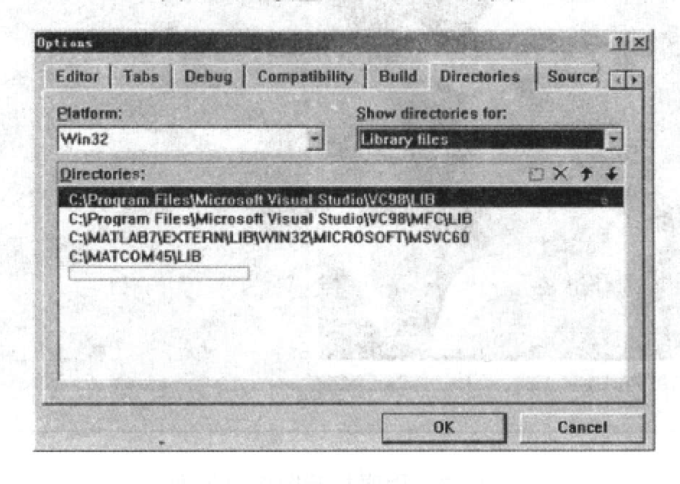


图 13-9　设置 Libray files 目录

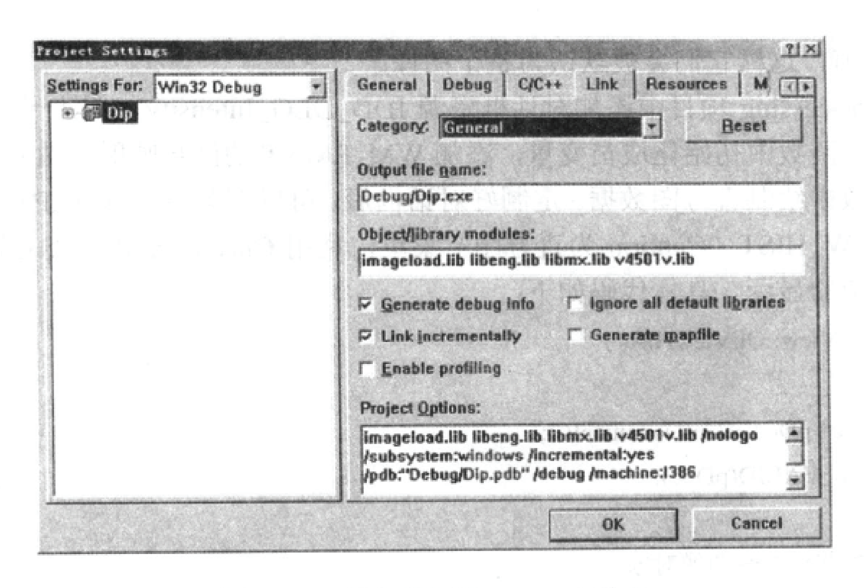


图 13-10　添加用到的 Libray 文件名

13.4.1　图像直方图统计——MATLAB 引擎命令实现

直方图分析是图像处理中一种常见的图像分析方法，描述了一幅图像的灰度分布情况，可用在图像分割、图像灰度变换等处理过程中。从概念上说，直方图是图像各灰度值的统计特性，即一幅图像中各个灰度级出现的次数或概率；从图形上说，它是一个二维图形，横坐标表示图像中各个像素点的灰度级，纵坐标为各个灰度级上图像各个像素点出现的次数或概率。如果不特别说明，直方图的纵坐标都对应着该灰度级在图像中出现的概率。

13.4.1.1　图像直方图性质

图像直方图有如下性质：

- 直方图是一幅图像中各像素灰度值出现次数（或频数）的统计结果，只反映该图像中不同灰度值出现的次数（或频数），而不能反映某一灰度值像素所在位置。也就是说，它只包含了该图像中某一灰度值的像素出现的概率，而丢失了其所在位置的信息。
- 任一幅图像，都能唯一地确定出一幅与它对应的直方图，但不同的图像可能有相同的直方图，即图像与直方图之间是多对一的映射关系。
- 由于直方图是对具有相同灰度值的像素进行统计得到的，因此，一幅图像各子区的直方图之和就等于该图像全图的直方图。

13.4.1.2　图像直方图统计 C++实现

直方图统计数据是通过 CPointPro:: CreateHistogram 函数实现的，函数定义如下：

```
int *CreateHistogram( int, int, int, int, unsigned char *, RGBQUAD *, int, CDibObject *pDibObject =
        NULL );
```

该函数创建传入的 CDibObject 对象中图像的直方图。如果进行此调整之前没有指定一个 CDibObject 对象指针，则必须在调整时加以指定。任何未传入的坐标值或默认的-1 坐标值都将被置为图像的最大值或最小值。变量 nX1 和 nY1 将被置为 0，nX2 将被置为图像宽度减 1，nY2 将被置为图像高度减 1。要在整个图像上进行操作时，最好的方法是不传入 nX1、nY1、

nX2 和 nY2 的值，这样它们会被默认为整个图像。

在 ImageProcessing 项目中添加对话框资源 IDD_DLG_Intensity，创建一个 CDlgIntensity 类。在它的构造函数中初始化成员变量，添加 WM_PAINT 的消息映射函数 OnPaint( )，然后在消息映射函数中绘制直方图数据。定制好对话框后，可以在菜单 IDR_DIPTYPE 的 View 项下添加 ID_VIEW_HIST（Caption 为直方图）菜单，利用 ClassWizard 在类 CdipView 中添加函数实现直方图的显示。具体代码如下：

```
void CDipView::OnViewHist( )
{
  CDipDoc* pDoc = GetDocument( );
  ASSERT_VALID(pDoc);

  //判断当前是否有图像对象
  if( pDoc->m_pDibObject == NULL ) return;

  //在点处理 CPointPro 类中创建用来绘制直方图的数据
  CPointPro PointOperation( pDoc->m_pDibObject );

  int *pHistogram = PointOperation.GetHistogram( );

  //生成一个对话框 CHistDlg 类的实例
  CDlgIntensity HistDlg;

  //将绘制直方图的数据传递给 CHistDlg 对话框类的公有成员变量 m_pnHistogram
  if( pHistogram != NULL )
  {
        //设置直方图数据指针
        HistDlg.m_pnHistogram = pHistogram;
        //设置当前像素值为 0 的像素数
        HistDlg.m_nCurrentPiexsNum = pHistogram[0];
        //设置是否为 256 级灰度图像
        HistDlg.m_bIsGray256 = PointOperation.IsGray256( );
  }

  //显示对话框
  if ( HistDlg.DoModal( ) != IDOK)
        return;

  delete [] pHistogram;
}
```

13.4.1.3 图像直方图统计混合编程实现——MATLAB 引擎命令实现

本节将通过 MATLAB 引擎实现图像的直方图统计，MATLAB 引擎库提供了一系列的函数使外部程序能和它进行交互。针对不同的命令组合，可以实现不同的混合编程方法。本节先介绍利用命令 engEvalString 的混合编程方法。

在 IDR_DIPTYPE 菜单资源的 View 菜单下添加 ID_VIEW_HISTMATLAB 菜单，利用 ClassWizard 在类 CDipdoc 添加响应函数，代码如下：

```
void CDipDoc::OnViewHistmatlab( )
{
    Engine *ep;     //定义 MATLAB 引擎变量
  CString command;
  if (!(ep=engOpen("\0")))   //打开 MATLAB 引擎
  {
        fprintf(stderr, "\n MATLAB 引擎启动失败!\n");
        MessageBox(NULL,"MATLAB 引擎启动失败!","MATLAB",MB_OK |MB_ICONERROR);
        exit(-1);
  }

  CString strPathName = GetPathName( );
  //实现 MATLAB 命令
  command="data=imread('"+strPathName+"')";

  //通过 MATLAB 引擎执行 MATLAB 命令
engEvalString(ep, command);
  engEvalString(ep, "figure,imshow(data)");
engEvalString(ep, "figure('Visible','off','NumberTitle','off', 'Name', '图像的直方图统计',
'MenuBar','none'), imhist(data)");

  MessageBox(NULL,"关闭 MATLAB 引擎，系统将退出 MATLAB 应用程序!","MATLAB",MB_OK |
MB_ICONINFORMATION);
  engClose(ep);  //关闭 MATLAB 引擎，退出 MATLAB*/
  }
```

13.4.1.4 图像直方图统计混合编程效果演示

编译整个项目文件，运行生成的 EXE 文件。通过“File”菜单下的“Open”子菜单，打开 image 目录下的 Miss.bmp 图像文件，然后在菜单“View”下选择“直方图”子菜单，可以看到利用 C++实现的直方图统计结果，如图 13-11 所示。在菜单“View”下选择“直方图－MATLAB 引擎”子菜单，程序将调用 MATLAB 引擎进行原始图像的显示及图像的直方图统计，结果分别如图 13-12 和图 13-13 所示。

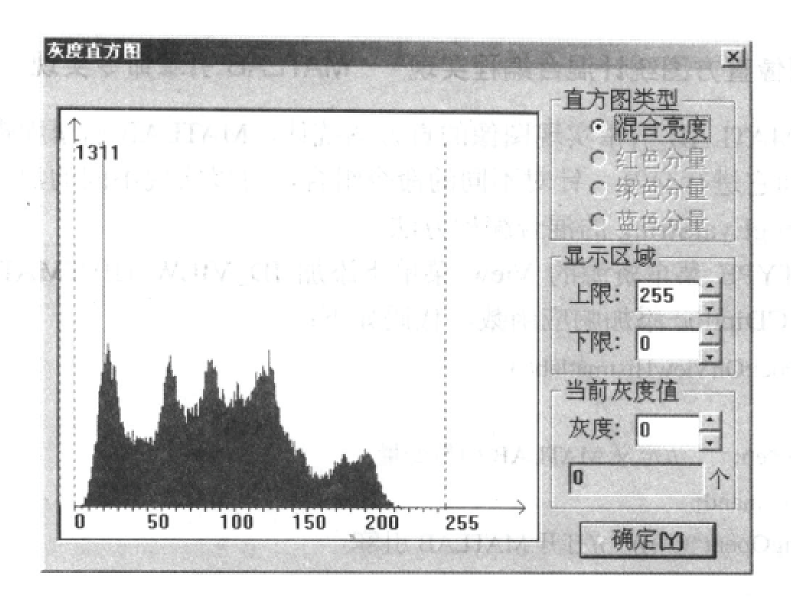


图 13-11　C++实现的直方图统计结果

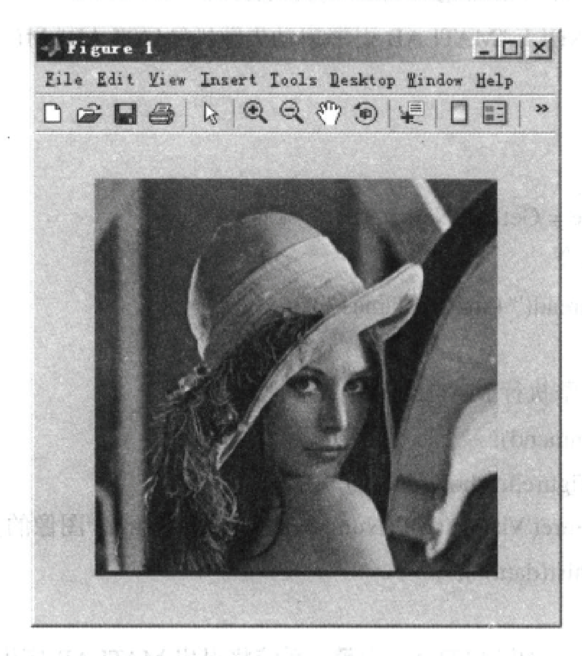


图 13-12　通过 MATLAB 引擎显示 Miss.bmp 文件

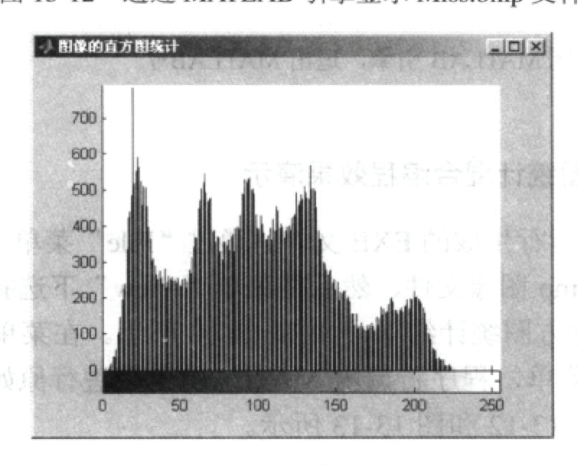


图 13-13　MATLAB 引擎实现直方图统计

13.4.2 图像形态学——MATLAB 引擎数据交互实现

数学形态学是一门建立在严格数学理论基础上的学科，主要是通过物体和结构元素相互作用的某些运算，获取物体拓扑和结构信息。数学形态学具有完备的数学基础，即集合论。

数学形态学的算法具有天然的并行实现的结构，可大大提高图像分析和处理的速度。它可用于简化图像数据，保持它们基本的形状特性，并除去不相关的结构。事实上，数学形态学已经构成一种新的图像处理方法和理论，成为计算机数字图像处理的一个重要研究领域，并且已经应用在多学科的数字图像分析和处理过程中。例如基于击中/击不中变换的目标识别，基于流域概念的图像分割，基于腐蚀和开运算的骨架抽取及图像编码压缩，基于测地距离的图像重建，基于形态学滤波器的颗粒分析等。

限于篇幅，本书只介绍简单二值图像的形态学运算，对于灰度图像的形态学运算，有兴趣的读者可以参阅相关的参考书。

13.4.2.1 腐蚀和膨胀

设 A 为图像集合，S 为结构元素，数学形态学运算是用 S 对 A 进行操作。需要指出，实际上结构元素本身也是一个图像集合。对每个结构元素可以指定一个原点，它是结构元素参与形态学运算的参考点。原点可以包含在结构元素中，也可以不包含在结构元素中，但运算的结果通常不相同。

二值图像基本的形态学运算是腐蚀和膨胀，其数学定义请参阅有关书籍。腐蚀是消除物体的所有边界点的一种过程，其结果是使剩下的物体沿其周边比原物体小一个像素的面积。如果物体是圆的，它的直径在每次腐蚀后将减少两个像素，如果物体在某一点处任意方向上连通的像素小于三个，那么该物体经过一次腐蚀后将在该点处分裂为两个物体。膨胀运算是将与某物体接触的所有背景点合并到该物体中的过程。过程的结果是使物体的面积增大了相应数量的点，如果物体是圆的，它的直径在每次膨胀后将增大两个像素。如果两个物体在某一点的任意方向相隔少于三个像素，它们将在该点连通起来。以下用阴影代表值为 1 的区域，白色代表值为 0 的区域，运算是对值为 1 的区域进行，如图 13-14 所示。

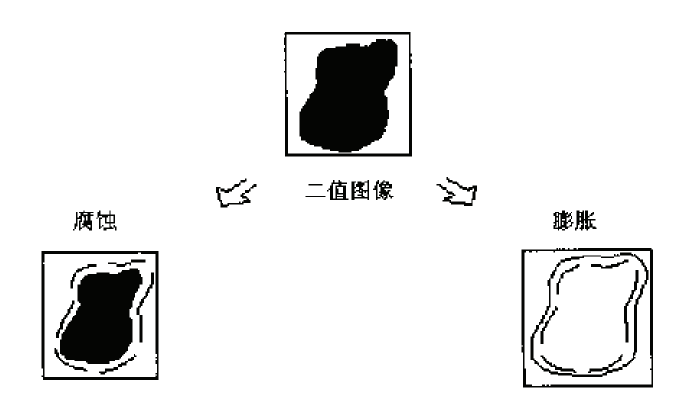


图 13-14 腐蚀和膨胀示意图

13.4.2.2 膨胀和腐蚀算法的 C++实现

膨胀和腐蚀算法的 C++实现代码如下：

```
////////////////////////////////////////////////////////////////////
```

```
//BOOL MakeErosion( )
//----------------------------------------------------------------------
//基本功能：本函数对图像数据执行腐蚀操作。
//----------------------------------------------------------------------
//参数说明：   int      *nMask              结构元素数组指针
//          int    nMaskLen              结构元素长度（以点数为计数单位）
//          unsigned char *pOut          输出图像数据指针
//          unsigned char *pIn           输入图像数据指针
//          int          nWidthBytes     图像宽度（以字节表示）
//          int          nWidth               图像宽度（以像素表示）
//          int          nHeight         图像高度
//----------------------------------------------------------------------
//返    回：BOOL
//          成功返回 TRUE，失败返回 FALSE。
//----------------------------------------------------------------------
//注    意：此函数声明为保护型，只能在 CMorphPro 类中使用。
//----------------------------------------------------------------------
////////////////////////////////////////////////////////////////////////
BOOL CMorphPro::MakeErosion(int *nMask, int nMaskLen,
                        unsigned char *pOut, unsigned char *pIn,
                        int nWidthBytes, int nWidth, int nHeight)
{
  //若传入的图像数据为空，将无法完成操作，直接返回
  if(pOut == NULL || pIn == NULL) return FALSE;

  //定义变量
  int x, y, k;
  unsigned char Mark;

  //执行腐蚀操作
  for( y = 0; y < nHeight; y++ )
  {
       unsigned char *pOutTemp = pOut;
       pOutTemp +=   y * nWidthBytes;
       for( x = 0; x < nWidth; x++ )
       {
            Mark = 1;
            for (k = 0; k < nMaskLen; k++)
            {
                 //不能处理边界像素
                 if ((x + nMask[2 * k] >= 0 ) &&
                      (x + nMask[2 * k] < nWidth) &&
```

```
                    (y + nMask[2 * k + 1] >= 0) &&
                    (y + nMask[2 * k + 1] < nHeight))
                {
                    unsigned char Data;
                    //取与模板中位置相对应的像素值
                    unsigned char *pTemp = pIn;
                    pTemp +=   y * nWidthBytes;
                    pTemp += nMask[2 * k + 1] * nWidthBytes;
                    Data = pTemp[x + nMask[2 * k]];
                    if(Data != 255)
                    {
                        Mark = 0;
                        k = nMaskLen;
                    }
                }
                else
                {
                    Mark = 0;
                    k = nMaskLen;
                }
            }
            if (Mark == 1)   pOutTemp[x] = 255;
        }
    }

    return TRUE;
}
```

函数 MakeErosion( )是一个保护型函数，在文档类中不能直接调用，CMorphPro 类（形态学处理类）提供了一个公有型函数 Erosion( )，可调用 MakeErosion( )进行腐蚀运算。有关 CmorphPro 类的详细说明请参考本书配套光盘。

Erosion( )函数核心代码如下：

```
////////////////////////////////////////////////////////////////////////
//BOOL Erosion( )
//----------------------------------------------------------------------
//基本功能：本函数对 CDibObject 对象中的图像进行腐蚀运算
//----------------------------------------------------------------------
//参数说明：int            *nMask         结构元素数组指针
//           int          nMaskLen       结构元素长度（以点数为计数单位）
//           CDibObject   *pDibObject    输出图像数据指针
//----------------------------------------------------------------------
//返    回：  成功返回 TRUE，失败返回 FALSE
```

```
//--------------------------------------------------------------------
////////////////////////////////////////////////////////////////////
BOOL CMorphPro::Erosion(int *nMask, int nMaskLen, CDibObject *pDibObject)
{
  //使用传入的 CDibObject 对象
  if( pDibObject != NULL ) m_pDibObject = pDibObject;
  //无 CDibObject 对象，返回 FALSE
  if( m_pDibObject == NULL ) return( FALSE );

  //不是 8 位灰度图像
  int nNumBits = m_pDibObject->GetNumBits( );
  if( nNumBits != 8 ) return( FALSE );

  //获得图像宽度和高度及字节宽度
  int nWidth = m_pDibObject->GetWidth( );
  int nHeight = m_pDibObject->GetHeight( );
  int nWidthBytes = m_pDibObject->WidthBytes(8, nWidth);

  //图像数据区大小
  DWORD dwSize = nWidthBytes * nHeight;

  //获得图像数据区指针
  unsigned char *pOldBuffer = GetBitsPoint( );

  //为新图像分配内存
  HGLOBAL hNewDib = ::GlobalAlloc( GMEM_MOVEABLE | GMEM_ZEROINIT, dwSize );

  if( hNewDib == NULL )
  {
       m_pDibObject->m_nLastError = IMAGELIB_MEMORY_ALLOCATION_ERROR;
       ::GlobalUnlock( m_pDibObject->GetDib( ) );
       return( FALSE );
  }

  //新图像数据指针
  unsigned char *pNewBuffer = (unsigned char *) ::GlobalLock( hNewDib );

     if (pNewBuffer == NULL || pOldBuffer == NULL)
  {
       AfxMessageBox("数据缓冲区定位出错！");
       return (FALSE);
  }
```

```
    //将原图像数据移动到新图像中（原图像数据清零）
    MoveBuffer(pNewBuffer, pOldBuffer, (LONG)dwSize);

    //调用 MakeErosion( )保护型函数进行腐蚀操作
    MakeErosion(nMask, nMaskLen, pOldBuffer, pNewBuffer, nWidthBytes, nWidth, nHeight);

    //将内存解锁和将不再使用的内存释放
    ::GlobalUnlock( m_pDibObject->GetDib( ) );
    ::GlobalUnlock( hNewDib );
    ::GlobalFree( hNewDib );
    return( TRUE );
}
```

当膨胀算法的 C++实现时，可仿照 MakeErosion( )函数编写一个 MakeDilation( )函数进行膨胀运算。与腐蚀类似，由 Dilation( )调用 MakeDilation( )函数来实现膨胀运算。

13.4.2.3 形态学应用实例一细化

下面利用二值形态学，给出一个进行二值区域形态学细化的实例。图像处理中常见的操作之一是“骨架化”，即对图像进行“细化”。图像细化是从原来的图像中去掉一些点，但仍要保持目标区域的原来形状，通过细化操作可以将一个物体细化为一条单像素宽的线，从而图形化地显示出其拓扑性质。实际上，图像细化就是保持原图的骨架。例如，一个长方形的骨架是它的长方向上的中轴线；正方形的骨架是它的中心点；圆的骨架是它的圆心，直线的骨架是它自身，孤立点的骨架也是自身。

图像处理中物体的形状信息是十分重要的，为了便于描述和抽取图像特定区域的特征，对那些表示物体的区域通常需要采用细化算法处理，得到与原来物体区域形状近似、由简单的弧或曲线组成的图形。在文字识别、地质构造识别、工业零件形状识别或图像理解中，先对被处理的图像进行细化有助于突出形状特点和减少冗余信息量。对于任意形状的区域，细化实质上是腐蚀操作的变体，细化过程中要根据每个像素点的八连通区域情况来判断该点是否可以剔除或保留。

细化算法的 C++实现是通过 CMorphPro::ThiningDIB( )实现的，函数定义如下：

```
////////////////////////////////////////////////////////////////////////
//BOOL ThiningDIB( )
//----------------------------------------------------------------------
//基本功能：本函数对 CDibObject 对象中的图像进行细化运算
//----------------------------------------------------------------------
//参数说明：CDibObject *pDibObject   默认为 NULL
//----------------------------------------------------------------------
//返    回：BOOL
//          成功返回 TRUE，失败返回 FALSE
//----------------------------------------------------------------------
```

```
////////////////////////////////////////////////////////////////////////
BOOL CMorphPro::ThiningDIB(CDibObject *pDibObject)
{
  //使用传入的 CDibObject 对象
  if( pDibObject != NULL ) m_pDibObject = pDibObject;
  //无 CDibObject 对象，返回 FALSE
  if( m_pDibObject == NULL ) return( FALSE );

  //定义变量
  unsigned char *pBuffer, *pBits;
  RGBQUAD *pPalette;
  int nWidthBytes, nNumColors;
  int lWidth,lHeight;

  //获得图像指针
  pBuffer = (unsigned char *) m_pDibObject->GetDIBPointer( &nWidthBytes,
 m_pDibObject->GetNumBits( ) );
  if( pBuffer == NULL ) return( NULL );
  //获得颜色数
  nNumColors = m_pDibObject->GetNumColors( );
  //获得调色板指针
  pPalette = (RGBQUAD *) &pBuffer[sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER)];
  //获得位图数据指针
  pBits = (unsigned char *) &pBuffer[sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER)+
nNumColors*sizeof(RGBQUAD)];

  lWidth=m_pDibObject->GetWidth( );
  lHeight=m_pDibObject->GetHeight( );
  ////////////////////////////////////////////////////////////
  // 指向源图像的指针
  LPSTR     lpSrc;
  // 指向缓存图像的指针
  LPSTR     lpDst;
  // 指向缓存 DIB 图像的指针
  LPSTR     lpNewDIBBits;
  HLOCAL  hNewDIBBits;
  //脏标记
  BOOL bModified;
  //循环变量
  long i;
  long j;
  int   n;
```

```
int   m;
//4 个条件
BOOL bCondition1;
BOOL bCondition2;
BOOL bCondition3;
BOOL bCondition4;
//计数器
unsigned char nCount;
//像素值
unsigned char pixel;
//5×5 相邻区域像素值
unsigned char neighbour[5][5];

// 暂时分配内存，以保存新图像
hNewDIBBits = LocalAlloc(LHND, lWidth * lHeight);
if (hNewDIBBits == NULL)
{
      // 分配内存失败
      return FALSE;
}
// 锁定内存
lpNewDIBBits = (char * )LocalLock(hNewDIBBits);
// 初始化新分配的内存，设定初始值为 255
lpDst = (char *)lpNewDIBBits;
memset(lpDst, (BYTE)255, lWidth * lHeight);
bModified=TRUE;
while(bModified)
{
      bModified = FALSE;
      // 初始化新分配的内存，设定初始值为 255
      lpDst = (char *)lpNewDIBBits;
      memset(lpDst, (BYTE)255, lWidth * lHeight);
      for(j = 2; j <lHeight-2; j++)
      {
            for(i = 2;i <lWidth-2; i++)
            {
                  bCondition1 = FALSE;
                  bCondition2 = FALSE;
                  bCondition3 = FALSE;
                  bCondition4 = FALSE;
                  //由于使用 5×5 的结构元素，为防止越界，所以不处理外围的几行和几列像素
```

```
// 指向源图像倒数第 j 行，第 i 个像素的指针
lpSrc = (char *)pBits + lWidth * j + i;
// 指向目标图像倒数第 j 行，第 i 个像素的指针
lpDst = (char *)lpNewDIBBits + lWidth * j + i;
//取得当前指针处的像素值，注意要转换为 unsigned char 型
pixel = (unsigned char)*lpSrc;
//目标图像中含有 0 和 255 外的其他灰度值
if(pixel != 255 && *lpSrc != 0)
	//return FALSE;
	continue;
//如果源图像中当前点为白色，则跳过
else if(pixel == 255)
	continue;
//获得当前点相邻的 5×5 区域内像素值，白色用 0 代表，黑色用 1 代表
for (m = 0;m < 5;m++ )
{
	for (n = 0;n < 5;n++)
	{
	neighbour[m][n] =(255 - (unsigned char)*(lpSrc + ((4 - m) - 2)*lWidth + n - 2 ))
	/ 255;
	}
}
//逐个判断条件
//判断 2<=NZ(P1)<=6
nCount =	neighbour[1][1] + neighbour[1][2] + neighbour[1][3] \
		+ neighbour[2][1] + neighbour[2][3] + \
		+ neighbour[3][1] + neighbour[3][2] + neighbour[3][3];
if ( nCount >= 2 && nCount <=6)
	bCondition1 = TRUE;
//判断 Z0(P1)=1
nCount = 0;
if (neighbour[1][2] == 0 && neighbour[1][1] == 1)
	nCount++;
if (neighbour[1][1] == 0 && neighbour[2][1] == 1)
	nCount++;
if (neighbour[2][1] == 0 && neighbour[3][1] == 1)
	nCount++;
if (neighbour[3][1] == 0 && neighbour[3][2] == 1)
	nCount++;
if (neighbour[3][2] == 0 && neighbour[3][3] == 1)
	nCount++;
if (neighbour[3][3] == 0 && neighbour[2][3] == 1)
```

```
        nCount++;
if (neighbour[2][3] == 0 && neighbour[1][3] == 1)
        nCount++;
if (neighbour[1][3] == 0 && neighbour[1][2] == 1)
        nCount++;
if (nCount == 1)
        bCondition2 = TRUE;

//判断 P2*P4*P8=0 or Z0(p2)!=1
if (neighbour[1][2]*neighbour[2][1]*neighbour[2][3] == 0)
        bCondition3 = TRUE;
else
{
        nCount = 0;
        if (neighbour[0][2] == 0 && neighbour[0][1] == 1)
                nCount++;
        if (neighbour[0][1] == 0 && neighbour[1][1] == 1)
                nCount++;
        if (neighbour[1][1] == 0 && neighbour[2][1] == 1)
                nCount++;
        if (neighbour[2][1] == 0 && neighbour[2][2] == 1)
                nCount++;
        if (neighbour[2][2] == 0 && neighbour[2][3] == 1)
                nCount++;
        if (neighbour[2][3] == 0 && neighbour[1][3] == 1)
                nCount++;
        if (neighbour[1][3] == 0 && neighbour[0][3] == 1)
                nCount++;
        if (neighbour[0][3] == 0 && neighbour[0][2] == 1)
                nCount++;
        if (nCount != 1)
                bCondition3 = TRUE;
}
//判断 P2*P4*P6=0 or Z0(p4)!=1
if (neighbour[1][2]*neighbour[2][1]*neighbour[3][2] == 0)
        bCondition4 = TRUE;
else
{
        nCount = 0;
        if (neighbour[1][1] == 0 && neighbour[1][0] == 1)
                nCount++;
        if (neighbour[1][0] == 0 && neighbour[2][0] == 1)
```

```
                nCount++;
            if (neighbour[2][0] == 0 && neighbour[3][0] == 1)
                nCount++;
            if (neighbour[3][0] == 0 && neighbour[3][1] == 1)
                nCount++;
            if (neighbour[3][1] == 0 && neighbour[3][2] == 1)
                nCount++;
            if (neighbour[3][2] == 0 && neighbour[2][2] == 1)
                nCount++;
            if (neighbour[2][2] == 0 && neighbour[1][2] == 1)
                nCount++;
            if (neighbour[1][2] == 0 && neighbour[1][1] == 1)
                nCount++;
            if (nCount != 1)
                bCondition4 = TRUE;
        }
        if(bCondition1 && bCondition2 && bCondition3 && bCondition4)
        {
            *lpDst = (unsigned char)255;
            bModified = TRUE;
        }
        else
        {
            *lpDst = (unsigned char)0;
        }
    }
  }
    // 复制腐蚀后的图像
    memcpy(pBits, lpNewDIBBits, lWidth * lHeight);
}
// 复制腐蚀后的图像
memcpy(pBits, lpNewDIBBits, lWidth * lHeight);
// 释放内存
LocalUnlock(hNewDIBBits);
LocalFree(hNewDIBBits);
// 返回
return TRUE;
}
```

在菜单 IDR_DIPTYPE 的形态学变换菜单项下添加 ID_MORPH_THINING（Caption 为细化）菜单，利用 ClassWizard 在类 CDipDoc 中添加函数实现图像的细化。具体代码如下：

```
void CDipDoc::OnMorphThining( )
{
```

```
    //创建形态学处理 CMorphPro 类对象
    CMorphPro MorphProcesses( m_pDibObject );

    // 更改光标形状
    BeginWaitCursor( );

    MorphProcesses.ThiningDIB(m_pDibObject);
    //恢复光标
    EndWaitCursor( );

    // 更新视图
    UpdateAllViews(NULL);
}
```

13.4.2.4 形态学算法的混合编程实现——MATLAB 引擎数据交互

在 ID 为“IDR_DIPTYPE”的“形态学变换”菜单项下添加子菜单，其标题为“MATLAB 形态学腐蚀”，ID 为“ID_MORPH_EROSION_MATLAB”。利用 ClassWizard 在类 CDipDoc 中添加函数 OnMorphErosionMATLAB( )实现图像的腐蚀。具体代码如下：

```
void CDipDoc::OnMorphErosionMATLAB( )
{
  //通过向 MATLAB 空间发送变量实现混合编程
     Engine *ep;      //定义 MATLAB 引擎变量
  mxArray *T = NULL;
  mxArray *Data= NULL;
  mxArray *Result=NULL;
  if (!(ep=engOpen("\0")))   //打开 MATLAB 引擎
  {
        fprintf(stderr, "\n MATLAB 引擎启动失败!\n");
        MessageBox(NULL,"MATLAB 引擎启动失败!","MaTLAB",MB_OK |MB_ICONERROR);
        exit(-1);
  }
  //向 MATLAB 空间传送图像文件名
  CString strPathName = GetPathName( );
  T=mxCreateString(strPathName);
  engPutVariable(ep, "T", T);

  //------------------------------------------
  //向 MATLAB 空间传送图像数据

  //图像指针为空，无法操作返回
  if(m_pDibObject == NULL) return;
```

```
//对不是 8 位的图像不作任何操作直接返回
if(m_pDibObject->GetNumBits( ) != 8)
{
    AfxMessageBox("目前只支持 8 位图像的处理！");
    return;
}

//获取图像宽度和高度（以像素为单位）
int nWidth = m_pDibObject->GetWidth( );
int nHeight = m_pDibObject->GetHeight( );

//定义变量
unsigned char *pBuffer;
unsigned char *pBits;
RGBQUAD *pPalette;
int nWidthBytes, nNumColors;

//获得图像指针
pBuffer = (unsigned char *) m_pDibObject->GetDIBPointer( &nWidthBytes,
                    m_pDibObject->GetNumBits( ));
if( pBuffer == NULL ) return;

//获得颜色数
nNumColors = m_pDibObject->GetNumColors( );
//获得调色板指针
pPalette = (RGBQUAD *) &pBuffer[sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER)];
//获得位图数据指针
pBits = (unsigned char *) &pBuffer[sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER)
                    + nNumColors * sizeof(RGBQUAD)];
//图像数据区大小（字节总数）
DWORD SizeImage = nWidthBytes * nHeight;
//-------------------------------------------------------

//创建图像数据矩阵
Data=mxCreateNumericMatrix(nHeight, nWidthBytes, mxUINT8_CLASS, mxREAL);
//用 memcpy 内存拷贝命令将图像数据赋予矩阵
    memcpy((char *)(mxGetPr(Data)), (char *)pBits, SizeImage);
//传送到 MATLAB 空间
engPutVariable(ep, "Data", Data);
//再用 rot90( )函数将二维矩阵逆时针旋转 90 度，将矩阵变为 nHeight×nWidthBytes 的二维矩阵
//并使的矩阵的第 nHeight 行对应图像数据的第一行数据
engEvalString(ep,"Data=rot90(Data)");
```

```
    engEvalString(ep,"SE = strel('square',[3 3])");
      engEvalString(ep,"Result = imerode(Data,SE)");
    engEvalString(ep,"Result=rot90(Result,-1)");
    Result=engGetVariable(ep,"Result");
    memcpy((char *)pBits, (char *)(mxGetPr(Result)), SizeImage);

    //关闭 MATLAB 引擎，退出 MATLAB*/
    engClose(ep);

      //内存解锁
      ::GlobalUnlock(m_pDibObject->GetDib( ));
      UpdateAllViews(NULL);
    }
```

类似地，在 ID 为“IDR_DIPTYPE”的“形态学变换”菜单项下添加“MATLAB 图像膨胀”和“MATLAB 图像细化”子菜单，并利用 ClassWizard 在类 CDipDoc 中添加对应的函数实现图像的膨胀和细化。图 13-15 至图 13-18 分别是 MATLAB 引擎实现图像 Circles.bmp 的腐蚀、膨胀和细化结果。

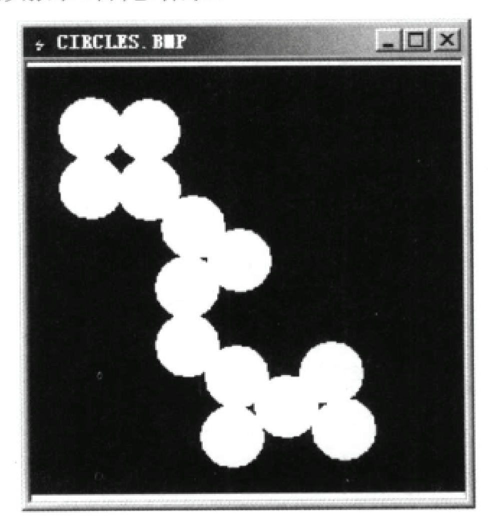


图 13-15　原始图像

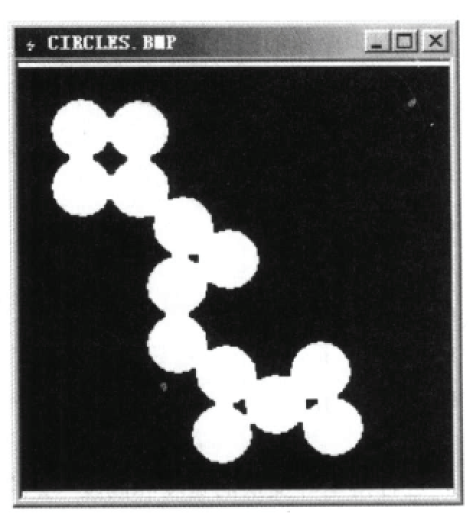


图 13-16　图像的腐蚀结果

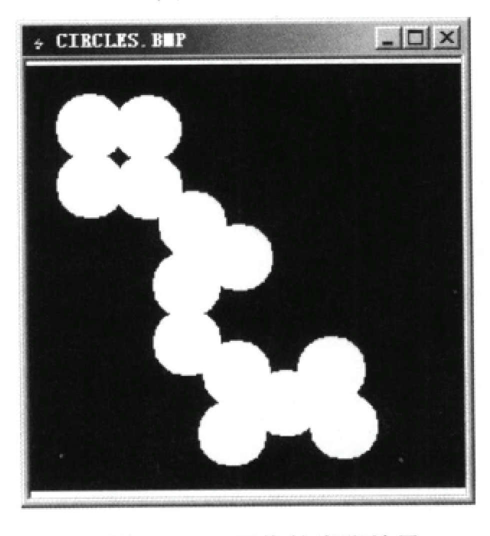


图 13-17　图像的膨胀结果

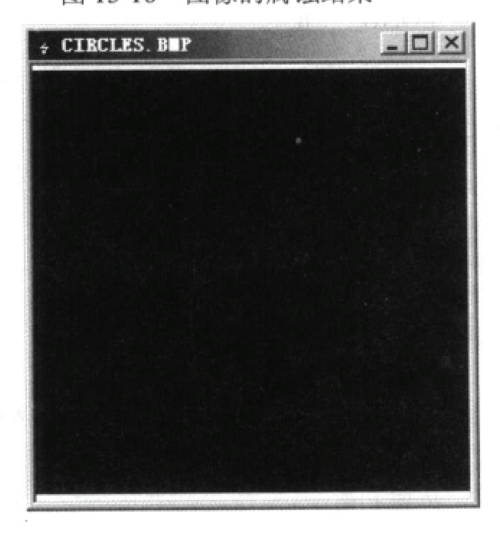


图 13-18　图像的细化结果

13.4.3 图像的 FFT 变换——通过 Mideva 实现

13.4.3.1 快速傅里叶变换介绍

对图像进行傅里叶变换，可以得到图像在频域的信息。对图像进行傅里叶变换还需要解决两个问题：一是在数学中进行傅里叶变换的 $f(x)$为连续（模拟）信号，而计算机处理的是数字信号（图像数据）；二是数学上采用无穷大概念，而计算机只能进行有限次计算。通常，将受这种限制的傅里叶变换称为离散傅里叶变换（Discrete Fourier Transform，DFT）。离散傅里叶变换计算量非常大，运算时间长，可以证明其运算次数正比于 N^2，特别是当 N 较大时，其运算时间将迅速增长，以至于无法接受。为此，研究离散傅里叶变换的快速算法（Fast Fourier Transform，FFT）是非常有必要的。

13.4.3.2 快速傅里叶变换的 C++实现

下面的代码用 C++实现了图像的二维快速傅里叶变换：

```
Fourier(LPSTR lpDIBBits, LONG lWidth, LONG lHeight)
{

 // 指向源图像的指针
 unsigned char*  lpSrc;

 // 中间变量
 double     dTemp;

 // 循环变量
 LONG    i;
 LONG    j;

 // 进行傅里叶变换的宽度和高度（2 的整数次方）
 LONG    w;
 LONG    h;

 int     wp;
 int     hp;

 // 图像每行的字节数
 LONG    lLineBytes;

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);

 // 赋初值
 w = 1;
```

```
h = 1;
wp = 0;
hp = 0;

// 计算进行傅里叶变换的宽度和高度（2 的整数次方）
while(w * 2 <= lWidth)
{
	w *= 2;
	wp++;
}

while(h * 2 <= lHeight)
{
	h *= 2;
	hp++;
}

// 分配内存
complex<double> *TD = new complex<double>[w * h];
complex<double> *FD = new complex<double>[w * h];

// 行
for(i = 0; i < h; i++)
{
	// 列
	for(j = 0; j < w; j++)
	{
		// 指向 DIB 第 i 行，第 j 个像素的指针
		lpSrc = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i) + j;

		// 给时域赋值
		TD[j + w * i] = complex<double>(*(lpSrc), 0);
	}
}

for(i = 0; i < h; i++)
{
	// 对 y 方向进行快速傅里叶变换
	FFT(&TD[w * i], &FD[w * i], wp);
}

// 保存变换结果
```

```
for(i = 0; i < h; i++)
{
	for(j = 0; j < w; j++)
	{
		TD[i + h * j] = FD[j + w * i];
	}
}

for(i = 0; i < w; i++)
{
	// 对 x 方向进行快速傅里叶变换
	FFT(&TD[i * h], &FD[i * h], hp);
}

// 行
for(i = 0; i < h; i++)
{
	// 列
	for(j = 0; j < w; j++)
	{
		// 计算频谱
		dTemp = sqrt(FD[j * h + i].real( ) * FD[j * h + i].real( ) +
				FD[j * h + i].imag( ) * FD[j * h + i].imag( )) / 100;

		// 判断是否超过 255
		if (dTemp > 255)
		{
			// 对于超过的，直接设置为 255
			dTemp = 255;
		}

		// 指向 DIB 第(i<h/2 ? i+h/2 : i-h/2)行，第(j<w/2 ? j+w/2 : j-w/2)个像素的指针
		// 此处不直接取 i 和 j，是为了将变换后的原点移到中心
		//lpSrc = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i) + j;
		lpSrc = (unsigned char*)lpDIBBits + lLineBytes *
			(lHeight - 1 - (i<h/2 ? i+h/2 : i-h/2)) + (j<w/2 ? j+w/2 : j-w/2);

		// 更新源图像
		* (lpSrc) = (BYTE)(dTemp);
	}
}
```

```
    // 删除临时变量
    delete TD;
    delete FD;

    // 返回
    return TRUE;
}
```

13.4.3.3 利用 Mideva 提供 Matrix<LIB>数学库实现快速傅里叶变换

要在 Visual C++开发环境下使用 Mideva 的 Matrix<LIB> C++数学库，必须在开发环境中进行一系列的设置。在调用 Matrix<LIB> C++库函数之前，首先要用函数 initM(Matcom_VERSION)来初始化类库调用，并用 exitM( )函数来结束类库调用，即应在相应的 CPP 文件中加入下列代码：

```
initM(Matcom_VERSION);
        //用 Matrix<LIB>C++库函数的调用完成矩阵操作
   exitM( );
```

当然可以在一个类的构造函数中添加 initM(Matcom_VERSION) 以完成类库的初始化工作。可在该类的析构函数中添加 exitM( )以完成结束类库调用的工作。

具体设置详见第 7 章。需要指出的是：如果开发的项目中同时用到了 MATLAB 引擎和 Mideva 的 Matrix<Lib> C++数学库，编译的时候会提示数据类型定义冲突。因此需要修改 Mideva 的 Matrix<Lib> C++数学库的头文件 matlib.h，修改后的头文件详见配书光盘。

在“频域变换”菜单项下添加 ID 为“ID_FFT_MIDEVA”，Caption 为“快速 FFT 变换”的子菜单，并利用 ClassWizard 在类 CDipDoc 中添加函数 OnFftMideva( )实现图像的快速傅里叶变换。具体代码如下：

```
void CDipDoc::OnFftMideva( )
{
    //初始化 Matrix<LIB>类库
    initM(Matcom_VERSION);

    Mm m_matBits;
    //读取图像数据
    //-----------------------------------------------
        //图像指针为空，无法操作返回
    if(m_pDibObject == NULL) return;

    //对不是 8 位的图像不作任何操作直接返回
    if(m_pDibObject->GetNumBits( ) != 8)
    {
        AfxMessageBox("目前只支持 8 位图像的处理！");
        return;
    }
```

```
//获取图像宽度和高度（以像素为单位）
int nWidth = m_pDibObject->GetWidth( );
int nHeight = m_pDibObject->GetHeight( );

//定义变量
unsigned char *pBuffer, *pBits;
RGBQUAD *pPalette;
int nWidthBytes, nNumColors;

//获得图像指针
pBuffer = (unsigned char *) m_pDibObject->GetDIBPointer( &nWidthBytes,
                                        m_pDibObject->GetNumBits( ));
if( pBuffer == NULL ) return;

//获得颜色数
nNumColors = m_pDibObject->GetNumColors( );
//获得调色板指针
pPalette = (RGBQUAD *) &pBuffer[sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER)];
//获得位图数据指针
pBits = (unsigned char *) &pBuffer[sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER)
                                + nNumColors * sizeof(RGBQUAD)];

//图像数据区大小（字节总数）
DWORD SizeImage = nWidthBytes * nHeight;

//创建图像数据矩阵，并将其元素初始值设为 0
m_matBits = zeros(1, SizeImage);

//默认的矩阵数据类型是 double，
//首先将其转换为 unsigned char 型，
//以便用 memcpy 内存拷贝命令将图像数据赋予矩阵
m_matBits = muint8(m_matBits);

//通过 Matrix<LIB>C++库的.addr( )函数返回矩阵变量的内存指针，以完成对矩阵单元的访问
//用 memcpy 内存拷贝命令将图像数据赋予矩阵
memcpy(m_matBits.addr( ), pBits, SizeImage);

//由于 Mm 类型的矩阵是按先列后行的顺序排列，
//在此用 reshape( )函数将创建的一维矩阵 m_matBits 变维为 nWidthBytes×nHeight 的二维矩阵
//再用 rot90( )函数将二维矩阵逆时针旋转 90°，将矩阵变为 nHeight×nWidthBytes 的二维矩阵，
//并使的矩阵的第 nHeight 行对应图像数据的第一行数据
```

```
m_matBits = rot90(reshape(m_matBits, nWidthBytes, nHeight));

//若图像宽度与其字节宽度不同，
//则将由系统补齐的每行字节数为 4 的整数倍的各列 0 删除，以减小矩阵大小加快处理速度
if(nWidthBytes != nWidth)
{
    //相当于 MATLAB 中的 X=X(:,1:nWidth)操作
    m_matBits = m_matBits(c_p, colon(1, 1, nWidth));
}

//将矩阵由 unsigned char 型转换为 double 型，以便进行运算
m_matBits = mdouble(m_matBits);

//内存解锁
::GlobalUnlock(m_pDibObject->GetDib( ));
//-------------------------------------------------------------
Mm mSize = size(m_matBits);
//调用 Matrix<Lib>C++库函数 fft2( )完成二维离散傅里叶变换
Mm ff = fft2(m_matBits);
Mm matTransed = ff;
//调用 Matrix<Lib>C++库函数 fftshift( )将频域中心移到矩阵中心
ff = fftshift(ff);
//调用 Matrix<Lib>C++库函数 mabs( )计算频谱
m_matBits = mabs(ff) / sqrt(mSize.r(1,1)*mSize.r(1,2));

//传回图像数据
//-------------------------------------------------------------

//将矩阵数据范围限定于（0——255）
m_matBits = mabs(m_matBits);
Mm L = m_matBits > 255.0;
Mm NotL = !L;
m_matBits = times(m_matBits, NotL) + L * 255.0;

//将矩阵由 double 型转换为 unsigned char 型，以便将图像数据赋予矩阵
m_matBits = muint8(m_matBits);

//补 0，以满足 BMP 图像对行宽字节数为 4 的整数倍的要求
int nTmp = (int)rem(nWidth, 4);
int nPadding;

if(nTmp > 0)
```

```
    {
        nPadding = 4 - nTmp;
        m_matBits = cat(2, m_matBits,
            repmat(muint8(0), (BR(size(m_matBits, 1)), nPadding)));
    }
    else
    {
        nPadding = 0;
    }

    //对矩阵进行转置操作
    m_matBits = rot90(m_matBits, -1);

    //将图像数据赋予矩阵
    memcpy(pBits, m_matBits.addr( ), (nWidthBytes * nHeight)*sizeof(unsigned char));

    //内存解锁
    ::GlobalUnlock(m_pDibObject->GetDib( ));
    //------------------------------------------------------------

    UpdateAllViews(NULL);

    //结束 Matrix<LIB>类库调用
    exitM( );
}
```

编译整个项目文件，运行生成的 EXE 文件，先打开 image 子目录下的 Miss.bmp 图像文件，如图 13-19 所示。然后单击“频域变换”菜单下“快速 FFT 变换”，可以实现图像的 FFT 变换，结果如图 13-20 所示。



图 13-19　打开的原始图像



图 13-20　图像的 FFT 变换结果

13.5 小　　结

本章综合了前面各章所学的知识，制作了一个小型的图像处理系统。图像处理功能中仅仅实现了一些简单的算法。希望有兴趣的读者，在该实例的基础上继续完善其功能，使其真正成为一款优秀的图像处理软件。读者可自由使用本光盘提供的所有资源，若发现 Bug 或对源代码有好的修改建议，请用电子邮件和作者联系。

附录A　常见的免费 MATLAB 工具箱

1．SDC Morphology Toolbox for MATLAB

数学形态学工具箱，在图像处理中应用广泛，包括图像分割、非线性滤波、模式识别和图像分析等内容。

http://www.mmorph.com/

2．Camera Calibration Toolbox for MATLAB

相机标定工具箱，可在 Windows、Unix 和 Linux 环境下运行。

http://www.vision.caltech.edu/bouguetj/calib_doc/

3．Speech Processing Toolbox for MATLAB

声音处理工具箱，包括声音文件输入/输出、声音合成、声音增强、声音分析、声音识别、声音编码等。

http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html

4．Statistical Pattern Recognition Toolbox for MATLAB

统计模式识别工具箱，包含用户手册和例程下载等内容。

http://cmp.felk.cvut.cz/~xfrancv/stprtool/

5．MATLAB and C Radar Toolbox

雷达信号处理工具箱。

http://www.radarworks.com/software.htm

6．FIR Filter Optimization Toolbox for MATLAB

FIR 滤波器优化工具箱。

http://www.csee.umbc.edu/~dschol2/opt.html

7．GPS Software Toolbox for MATLAB

GPS 工具箱。

http://www.navtechgps.com/supply/gps_toolbox.asp

8．Computer Vision Toolbox for MATLAB

计算机视觉工具箱。

http://www.icg.tu-graz.ac.at/research/ComputerVision/icg_visiontools

9．Dimensional Analysis Toolbox for MATLAB

多维分析工具箱。

http://www.sbrs.net/

附录 B 常用的 MATLAB 免费站点

国外站点：

- MathWorks 公司主页：http://www.mathworks.com
- MATLAB file exchange：http://www.mathworks.com/matlabcentral/fileexchange/
  世界各地的 MATLAB 编程爱好者提供的各种 MATLAB 函数，有很强的实用价值
- MATLAB 中心：http://www.mathworks.com/matlabcentral/
  包括了 MATLAB 新闻组、MATLAB 程序大赛与文件交换中心等页面
- 德国的一个 M 文件数据库：http://matlabdb.mathematik.uni-stuttgart.de/index.jsp5）
- MathTools 站点：http://www.mathtools.net/MATLAB/index.html
  有很多好的工具箱或者小的辅助函数

国内中文站点：

- 恒润科技：http://www.hirain.com/ MathWorks 公司的 MATLAB 国内独家代理商
- MATLAB 大观园：http://matlab.myrice.com/
- 中国学术交流网地：http://www.matwav.com/resource/newlk.asp
- 仿真论坛 MATLAB 版：http://www.simwe.com/cgi-bin/ut/board_show.cgi?id=19&ge=30
- 动力学控制论坛工程数学软件版：http://www.dytrol.com/index.asp
- 研学论坛 MATLAB 版：http://bbs.matwav.com/index.jsp
- MATLAB 语言及应用：http://sh.netsh.com/bbs/5186/
- 中科大 AIAM 数学工具论坛：http://mcm.ustc.edu.cn/forum/index.php

国内大学 BBS：

- 水木清华 MathTools 版：http://www.smth.org/
- 哈工大紫丁香 MATLAB 版：http://bbs.hit.edu.cn/
- 上海交大饮水思源 MathTools 版：http://bbs.sjtu.edu.cn
- 瀚海星云：http://fbbs.ustc.edu.cn/

参 考 文 献

1 苏金明，黄国明，刘波．MATLAB 与外部程序接口．北京：电子工业出版社，2004 年

2 刘维．精通 Matlab 与 C/C++混合程序设计．北京：北京航空航天大学出版社，2005 年

3 何强，何英．MATLAB 扩展编程．北京：清华大学出版社，2002 年

4 刘志俭等．MATLAB 应用程序接口用户指南．北京：科学出版社，2000 年

5 飞思科技产品研发中心．MATLAB 6.5 应用接口编程．北京：电子工业出版社，2003 年

6 马兴义，杨立群，林敏，龚少华．Matlab 6 应用开发指南．北京：机械工业出版社，2002 年

7 A. Riazi．MATLAB Engine API．http://www.codeproject.com/samples/matlabeng.asp, 2005-06-10

8 A. Riaz．Solving Engineering Problems Using MATLAB C++ Math Library．http://www.codeproject.com/samples/matlab_cpp.asp，2005-06-12

9 Cambalindo．Using MatLab Add-in for MS Visual Studio 6．http://www.codeproject.com/macro/using_matlab_add_in.asp，2005-06-14

10 A. Riaz．MATLAB MEX-files．http://www.codeproject.com/samples/mexFunction.asp，2005-06-20

11 A. Riaz．Building COM Components Using MATLAB．http://www.codeproject.com/script/profile/whos_who.asp?id=52350，2005-07-17